Kriging with Partial Differential Equations in Hydrogeology

Pierre Le Cointe

Université Pierre et Marie Curie, École des Mines de Paris
& École Nationale du Génie Rural des Eaux et des Forêts

Master 2 Sciences de l'Univers, Environnement, Ecologie
Parcours Hydrologie-Hydrogéologie

Kriging with Partial Differential Equations in Hydrogeology

Pierre LE COINTE

Directeur de recherche : Jean-Pierre DELHOMME

Schlumberger Water Services
Waterloo Hydrogeologic Inc.
Waterloo, Ontario, Canada

January 23, 2007

Abstract

Drawing hydraulic head contour maps is one of the first requested results in hydrogeology. This goal can be achieved in several manners : solving the partial differential equation within discretized models calibrated so as to fit the head data, or more simply using spatial data interpolation techniques. One of these techniques is kriging, a stochastic approach that estimates the value of a natural phenomenon in unsampled sites, using an unbiased linear combination of neighboring measures of the phenomenon, with a minimum variance. The first part of this thesis explains the basics of the kriging theory.

However, the main goal of this work is to combine this geostatistical method with some features of the partial differential equations problem to provide a mapping tool that has the kriging simplicity of use but provides scientifically better results. Two different improvements are described in this thesis. The first one introduces the use of boundary conditions in the kriging algorithm, and the second one focuses on covariance models that take into account the transmissivity values, both for kriging and cokriging estimation. For both features, the theoretical explanation is followed by application examples that highlight the improvements in the estimated results. Finally, the use of the improved cross-covariance models to solve the inverse problem (determining the transmissivity knowing the hydraulic head) is detailed. All the examples were made using GSLIB kriging and cokriging algorithms that were modified for that purpose.

The results presented in this thesis show that Kriging under Boundary Conditions is an efficient way of improving the interpolated contour maps without involving discretized modeling. Cokriging between the hydraulic head and the transmissivity with a structural analysis focused on ensuring that the two variables verify the partial differential equation allows to take into account the variations of the transmissivity while assuming a regionally monodirectional flow of constant hydraulic gradient J and a discharge $Q = 0$. It can be used to solve both the direct and the inverse problem. The first results are promising, but there is still some work to be done to have a tool as robust as Kriging under Boundary Conditions.

Keywords :

Geostatistics, kriging, cokriging, covariance, variogram, hydrogeology, hydraulic head, transmissivity, boundary conditions, prescribed head, prescribed flux, gradient information, interpolation, contour mapping, modeling, inverse problem.

Acknowledgements

I would like to thank here all the people that contributed more or less to this work. First of all, I am grateful to my tutor, Dr. Jean-Pierre Delhomme, SWS Scientific Advisor, who offered me this opportunity to combine hydrogeology with geostatistics and programming. I had some (little) experience in all these fields, but never had the opportunity to work on the three of them altogether. Jean-Pierre Delhomme obviously provided some invaluable help for this work, especically on the purely geostatistical part, as the guiding ideas were his.

I also owe special thanks to Dr. Benny Bian, Software Department Manager in Waterloo, and Dr. Serguei Chmakov, Software Chief Architect. Benny Bian was my manager during my stay in Canada and always showed great interest in my work. Serguei Chmakov was always very helpful whenever it came down to programming or hydrogeology issues. His fast understanding and scientific knowledge never left me in trouble for a long time.

Finally, I would like to thank Dominique Pajot, Marketing and Technique Manager of Schlumberger Water Services, for hiring me, Sandra Jenni of SWS Paris, for the last minute help, as well as all the other employees of SWS from Waterloo, Paris or Calgary that helped me with technical or administrative issues before and during my stay in Canada. My last word will be for my schoolmates Lionel, Mathilde and Thomas that convinced me to switch to LaTeX to write this report. Their advice probably made the result much more readable !

Contents

List of Figures

Introduction

One of the main parameters hydrogeologists need to know for their studies is the hydraulic head. Thus, drawing adequate head contour maps is a common issue in hydrogeology. Two very different methodologies can be used to achieve this goal :

- **Solving the partial differential equation within discretized models :**
  This method requires the input of the needed parameters to solve the diffusivity equation so that the computed head surface knows the head data points. In most cases, the problem is too complex to find an analytical solution and a numerical simulation has to be computed, often based on either the finite difference or finite element methods, as explained by *de Marsily* (1981). This approach provides the best results, but it certainly requires some knowledge in hydrogeologic modeling, sufficient parameters data, and, obviously, the adequate code.

- **Using a spatial interpolation technique :**
  Spatial interpolation is a mathematical processing that allows the reconstruction of a phenomenon over a domain based on a limited number of data samples of this phenomenon. Basically, one only has to enter his data and choose an interpolation method to build his contour map. However, if the estimate produced is correct, it does not verifiy the same partial differential equation that the real data does. In hydrogeology, that means that the flow equation will not be verified.

Knowing this, the goal of this thesis is to suggest some elements to combine both of these very different methods to produce hydraulic head maps that are scientifically better than the ones obtained with classic interpolation, but still easier to create than the ones made by solving the partial differential equations. The chosen interpolator is kriging, because it already takes into account the spatial dependency of the data, and the programming work has been centered on Geostatistical Software Library (GSLIB) kriging and cokriging algorithms. GSLIB is available in the public domain, distributed by Stanford University and documented in *Deutsch and Journel* (1998). Their algorithms are widely used in research or commercial codes. In particular, they are present in several Waterloo Hydrogeologic software programs, including *GW Contour*, an easy-to-use data interpolation and contouring program that also provides techniques for mapping velocity vectors and particle tracks[1]. The aim of my internship was to improve the kriging algorithm for this software, in order to have better head maps, and thus better velocity vectors and particle tracking.

The first chapter of this thesis presents the kriging theory, which was the interpolation method chosen to implement the new features. The second chapter explains how kriging

[1]More information available on http://www.waterloohydrogeologic.com/software/gw_contour/gw_con\discretionary{-}{}{}tour_ov.htm

can take care of the boundary conditions when mapping the hydraulic head, and presents some examples that show the improvements resulting from this addition. The third chapter suggests another way of improving the head kriging while using some transmissivity data and the partial differential equation background to improve the covariance computation. It also details the cokriging process between head and transmissivity, while also taking into account the diffusivity equation results to compute the cross-covariance. This method can actually be used for both estimating the hydraulic head and the transmissivity, thus in theory, it can be used to solve the inverse problem. Finally, the conclusion rounds up the results of these various researches, explains how they will be implemented in *GW Contour*, and what the further developments could be.

Chapter 1

Kriging

The word kriging and the method itself have been created by G. Matheron in 1963, after the name of D.G. Krige, a South-African mining engineer whose work initiated Matheron's. This chapter presents some elements of the theory of regionalized variables needed to understand the kriging method. The mathematic process is then described for Universal Kriging, also called Kriging with a Trend model, which is the most "generalized" version of kriging. Finally, some characteristics useful for the following chapters are detailed. All the theory presented in this chapter comes from the work of *Matheron* (1962, 1963, 1965, 1969, 1970, 1971a, 1973, 1974). The following references have also been helpful : *Geostatistics : Modeling Spatial Uncertainty* by *Chilès and Delfiner* (1999), *GSLIB : Geostatistical Software Library and User's Guide* by *Deutsch and Journel* (1998), the various lecture notes on Geostatistics from the Ecole Nationale Supérieure des Mines de Paris by *Chauvet* (1993); *Rivoirard* (1995, 2003); *Wackernagel* (1993) and *Le krigeage : revue de la théorie et application à l'interpolation spatiale de données de précipitation*, a well done Master's Thesis by *Baillargeon* (2005).

1.1 Prerequisites

1.1.1 Random Variable

The basic paradigm of predictive statistics is to characterize any unknown value z as a random variable (RV) Z, the probability distribution of which models the uncertainty about z. A random variable is a variable that can take a variety of outcome values according to some probability distribution. The RV model Z, and more specifically its probability distribution, is usually location-dependent ; hence the notation $Z(x)$, with x being the location coordinates vector of a point. In the continuation of this thesis, we will work in $\mathbb{R}^2$, thus a point x will be defined by its two coordinates `x` and `y`. The RV $Z(x)$ is also information-dependent in the sense that its probability distribution changes as more data about the unsampled value $z(x)$ become available.

The cumulative distribution function (cdf) of a continuous RV $Z(x)$ is denoted :

$$F(x;z) = \text{Prob}\left\{Z(x) \leq z\right\} \tag{1.1}$$

When the cdf is made specific to a particular information set, for example (n) consisting of n neighboring data values $Z(x_\alpha) = z(x_\alpha), \alpha = 1, \ldots, n$, the notation "conditional to n" is used,

defining the conditional cumulative distribution function (ccdf) :

$$F(x;z|(n)) = \text{Prob}\,\{Z(x) \leq z|(n)\} \tag{1.2}$$

Expression (1.1) models the uncertainty about the unsampled value $z(x)$ prior to using the information set (n) while expression (1.2) models the posterior uncertainty once the information set (n) has been accounted for. The goal of any predictive algorithm is to update prior models of uncertainty such as (1.1) into posterior models such as (1.2). The ccdf $F(x;z|(n))$ is a function of the location x, the sample size and geometric configuration (i.e. the data locations $x_\alpha, \alpha = 1,\ldots,n$), and the sample values $z(x_\alpha), \alpha = 1,\ldots,n$.

From the ccdf (1.2) one can derive different optimal estimates for the unsampled value $z(x)$ in addition to the ccdf mean, which is the least-squares error estimate. One can also derive various probability intervals.

In geostatistics, most of the information related to an unsampled value $z(x)$ comes from sample values at neighboring locations x', whether defined on the same attribute z or on some related attribute y. Thus it is important to model the degree of correlation or dependence between any number of RVs $Z(x), Z(x_\alpha), \alpha = 1,\ldots,n$ and more generally $Z(x), Z(x_\alpha), \alpha = 1,\ldots,n, Y(x'_\beta), \beta = 1,\ldots,n'$. The concept of a random function (RF) allows such modeling and updating of prior cdfs into posterior ccdfs.

1.1.2 Random Function

A random function (RF) is a set of RVs defined over some field of interest, such as $\{Z(x), x \in \text{study area}\}$ also denoted simply as $Z(x)$. Usually the RF definition is restricted to RVs related to the same attribute, say z, hence another RF would be defined to model the spatial variability of a second attribute, say $\{Y(x), x \in \text{study area}\}$.

Just as an RV $Z(x)$ is characterized by its cdf (1.1), an RF $Z(x)$ is characterized by the set of all its K-variate cdfs for any number K and any choice of the K locations $x_k, k = 1,\ldots,K$:

$$F(x_1,\ldots,x_K;z_1,\ldots,z_K) = \text{Prob}\,\{Z(x_1) \leq z_1,\ldots,Z(x_K) \leq z_K\} \tag{1.3}$$

Just as the univariate cdf of RV $Z(x)$ is used to model uncertainty about the value $z(x)$, the multivariate cdf (1.3) is used to model joint uncertainty about the K values $z(x_1),\ldots,z(x_K)$.

Of particular interest is the bivariate $(K = 2)$ cdf of any two RVs $Z(x)$, $Z(x')$, or more generally $Z(x)$, $Y(x')$:

$$F(x,x';z,z') = \text{Prob}\,\{Z(x) \leq z, Y(x') \leq z'\} \tag{1.4}$$

1.1.3 Other Definitions

1.1.3.1 Regionalized Variable

G. Matheron defined a regionalized phenomenon as a phenomenon that spreads in space and exhibits a certain spatial structure. If $z(x)$ denotes the value at the point $x \in D$ of a characteristic z of this phenomenon, we shall say that $\{z(x) : x \in D \subset \mathbb{R}^n\}$ is a regionalized variable.

The key, in geostatistics, is that we act *as though* the regionalized variable under study $z(x)$ is a realization of a parent random function $\{Z(x) : x \in \mathbb{R}^n\}$. In particular, *Delhomme* (1976, 1978) demonstrated that a number of fields of hydrogeologic variables (head, transmissivity, thickness of aquifer layers...) possess a spatial structure and are therefore amenable to geostatistical techniques.

1.1.3.2 Spatial Distribution

A random function is described by its finite-dimensional distributions, namely the set of all multidimensional distributions of K-tuples $(Z(x_1), \ldots, Z(x_K))$ for all finite values of K and all configurations of the points $x_1, \ldots, x_K$. That is what we call the spatial distribution.

1.1.3.3 Distance between two points

The distance between two points in $\mathbb{R}^2$, $A(x_A, y_A)$ and $B(x_B, y_B)$, is defined by the Euclidean norm of the vector determined by these two points :

$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

1.1.3.4 Moments

The mean of the RF is the expected value $m(x) = E[Z(x)]$ of the RV $Z(x)$ at point x. It is also called the drift of Z, especially when $m(x)$ varies with location. The (centered) covariance $Cov(x, y)$ is the covariance of the RV $Z(x)$ and $Z(y)$:

$$Cov(x, y) = E\Big[\big(Z(x) - m(x)\big)\big(Z(y) - m(y)\big)\Big] \tag{1.5}$$

In general, this function depends on both x and y. When $x = y$, $Cov(x, x) = Var[Z(x)]$ is the variance of $Z(x)$. Higher-order moments can be defined similarly.

Naturally, in theory, these moments may not exist. As usual in probability theory the mean is defined only if $E\,|Z(x)| < \infty$. If $E[Z(x)]^2$ is finite at every point, $Z(x)$ is said to be a second-order RF : it has a finite variance, and the covariance exists everywhere.

1.1.4 Stationarity Hypothesis

1.1.4.1 Strict Stationarity

A RF is called stationary when the finite-dimensional distributions are invariant under an arbitrary translation of the points by a vector h :

$$\text{Prob}\,\{Z(x_1) < z_1, \ldots, Z(x_K) < z_K\} = \text{Prob}\,\{Z(x_1 + h) < z_1, \ldots, Z(x_K + h) < z_K\} \tag{1.6}$$

Physically, this means that the phenomenon is homogeneous in space and repeats itself in the whole space.

1.1.4.2 Second-Order Stationarity

When the random function is stationary, its moments, if they exist, are obviously invariant under translations. The second-order stationarity hypothesis consider that the first two moments (mean and covariance) are stationary. We have then for points x and $x+h$ of $\mathbb{R}^n$:

$$\begin{cases} E\big[Z(x)\big] = m \\ E\Big[\big(Z(x)-m\big)\big(Z(x+h)-m\big)\Big] = E\big[Z(x)Z(x+h)\big] - m^2 = C(h) \end{cases} \tag{1.7}$$

The mean is constant and the covariance function C has the following properties :

- It only depends on the separation h,
- It is bounded and doesn't exceed the constant variance :

$$|C(h)| \leq C(0) = Var\big(Z(x)\big)$$

- It is an even function : $C(-h) = C(h)$.

By definition, an RF satisfying the above conditions is second-order stationary and will be further called Stationary Random Function or SRF. An SRF is isotropic if its covariance function only depends on the length $|h|$ of the vector h, and not on its orientation.

1.1.4.3 Intrinsic Hypothesis

A milder hypothesis is to assume that for every vector h the increment $Y_h(x) = Z(x+h) - Z(x)$ is an SRF in x. Then $Z(x)$ is called an intrinsic random function (IRF) and is characterized by the following relationships :

$$\begin{cases} E\big[Z(x+h) - Z(x)\big] = \langle a, h\rangle \\ E\Big[\big(Z(x+h) - Z(x)\big)^2\Big] = Var\big[Z(x+h) - Z(x)\big] = 2\,\gamma(h) \end{cases} \tag{1.8}$$

$\langle a, h\rangle$ is the linear drift of the IRF (drift of the increment) and $\gamma(h)$ is its variogram function.

If the linear drift is zero, that is, if the mean is constant, we have the usual form of the intrinsic model :

$$\begin{cases} E\big[Z(x+h) - Z(x)\big] = 0 \\ Var\big[Z(x+h) - Z(x)\big] = 2\,\gamma(h) \end{cases} \tag{1.9}$$

This gives us a definition of the usual form of the theoretical variogram :

$$\gamma(h) = \frac{1}{2} E\Big[\big(Z(x+h) - Z(x)\big)^2\Big] \tag{1.10}$$

The variogram has the following properties :

- It only depends on the separation h,
- It is an even function : $\gamma(-h) = \gamma(h)$,

- It is nonnegative : $\gamma(h) \geq 0$, and $\gamma(h = 0) = 0$.

Existence of the expectation of the increments of an IRF does not imply the existence of the expectation of the IRF itself. An IRF can have an infinite variance while its increments do have a finite variance for each vector h. In particular, that means that, whereas we can infer the variogram from the covariance function with the following formula :

$$\gamma(h) = C(0) - C(h), \quad \forall h \tag{1.11}$$

the opposite is not true : you can't define a covariance function from every variogram.

1.1.5 A Useful Result

The following calculation provides another result that links the covariance and the variogram. It will be used further to establish the kriging system. When $E\big[Z(x_i) - Z(x_0)\big] = E\big[Z(x_j) - Z(x_0)\big] = 0$,

$$\begin{aligned}
&Cov\left[Z(x_i) - Z(x_0)\,,\, Z(x_j) - Z(x_0)\right] \quad = \\
&= E\Big[\big(Z(x_i) - Z(x_0)\big)\big(Z(x_j) - Z(x_0)\big)\Big] \\
&= E\Big[Z(x_i)Z(x_j) - Z(x_i)Z(x_0) - Z(x_j)Z(x_0) + Z(x_0)^2\Big] \\
&= E\big[Z(x_i)Z(x_j)\big] - E\big[Z(x_i)Z(x_0)\big] - E\big[Z(x_j)Z(x_0)\big] + E\big[Z(x_0)^2\big] \\
&= \left(-\frac{1}{2}E\big[Z(x_i)^2\big] + E\big[Z(x_i)Z(x_j)\big] - \frac{1}{2}E\big[Z(x_j)^2\big]\right) \\
&\qquad + \left(\frac{1}{2}E\big[Z(x_i)^2\big] - E\big[Z(x_i)Z(x_0)\big] + \frac{1}{2}E\big[Z(x_0)^2\big]\right) \\
&\qquad + \left(\frac{1}{2}E\big[Z(x_j)^2\big] - E\big[Z(x_j)Z(x_0)\big] + \frac{1}{2}E\big[Z(x_0)^2\big]\right) \\
&= -\frac{1}{2}E\big[Z(x_i) - Z(x_j)\big]^2 + \frac{1}{2}E\big[Z(x_i) - Z(x_0)\big]^2 + \frac{1}{2}E\big[Z(x_j) - Z(x_0)\big]^2 \\
&= -\gamma(x_i - x_j) + \gamma(x_i - x_0) + \gamma(x_j - x_0)
\end{aligned}$$

This result :

$$Cov\big[Z(x_i) - Z(x_0)\,,\, Z(x_j) - Z(x_0)\big] = -\gamma(x_i - x_j) + \gamma(x_i - x_0) + \gamma(x_j - x_0) \tag{1.12}$$

combined with equation (1.11) provides, if the covariance function is defined :

$$Cov\big[Z(x_i) - Z(x_0)\,,\, Z(x_j) - Z(x_0)\big] = C(x_i - x_j) - C(x_i - x_0) - C(x_j - x_0) + C(0) \tag{1.13}$$

More precisely, the result that will be further used to solve the kriging system is :

$$E\Big[\big(Z(x_i) - Z(x_0)\big)\big(Z(x_j) - Z(x_0)\big)\Big] = C(x_i - x_j) - C(x_i - x_0) - C(x_j - x_0) + C(0) \tag{1.14}$$

1.2 Structural Analysis

1.2.1 Experimental Variogram

In practice, the spatial variability of a regionalized variable $z(x)$ can be measured at various scales by computing the difference between two data values z_1 and z_2 located in two points x_1 and x_2 of the spatial distribution. This difference $\gamma^\star$ is defined by :

$$\gamma^\star = \frac{(z_2 - z_1)^2}{2}$$

$\gamma^\star$ depends on the distance and the orientation of the pair of points, described by the vector $h = x_2 - x_1$, whatever the position of the points in the spatial distribution is :

$$\gamma^\star(h) = \frac{1}{2}\Big(z(x_1 + h) - z(x_1)\Big)^2$$

Taking the mean of the $\gamma^\star$ differences for all the n_h couples of data points linked by a given vector h for a given mesh, we can build the experimental variogram :

$$\gamma^\star(h) = \frac{1}{2}\sum_{\alpha=1}^{n_h}\Big(z(x_\alpha + h) - z(x_\alpha)\Big)^2 \tag{1.15}$$

1.2.2 Variogram Characteristics

1.2.2.1 Nugget Effect

The behavior of the variogram near its origin (i.e. for small values of h) is key, as it shows the degree of continuity of the regionalized variable : differentiable, continuous but not differentiable, or not continuous. If this last case is true, i.e. if $\lim_{h\to 0^+}\gamma(h) = C_0 > 0$, then C_0 is called the nugget effect (see figure 1.1). A steep nugget effect denotes a weak correlation between two very close data values. This can be explained by some undetected variations at a very small scale. The name "nugget effect" has been given after the fact that such big variations at a small scale can be observed in gold deposits, where there are gold nuggets.

1.2.2.2 Sill and Range

Usually, we tend to notice that $\gamma^\star(h)$ increases with $|h|$ and it frequently reaches a variation plateau for big distances. When this plateau is reached, that means that there is no further spatial dependency between data. This distance is called "range", and the word "sill" describes the variance for which this plateau appears (see figure 1.1). Sometimes, the sill is only reached asymptotically. In that case, the real range in infinite, but a practical range is defined by the distance at which the variogram reaches 95% of the value of its sill.

If a variogram is not bounded, it does not have any range nor sill. The variance of the RF is then undefined, and such an RF is not an SRF, but only an IRF. Another possibility is to notice that the variogram slope changes steeply. One can then imagine that there is an intermediate sill. That in fact means that the variogram has several nested structures, each one being defined by its own range and sill (see figure 1.1).

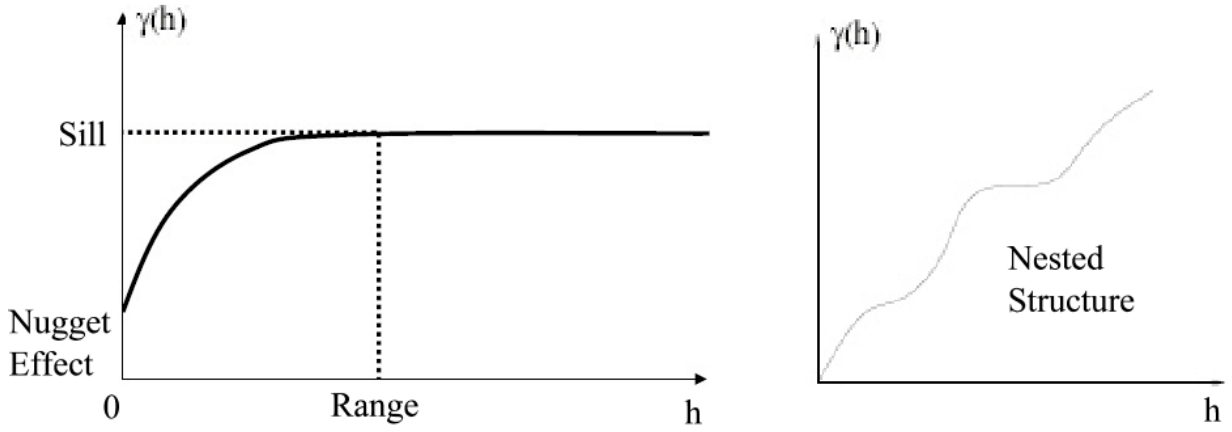


Figure 1.1: A variogram and a nested structure example.

1.2.3 Variogram Examples

The goal of this thesis is not to explain how one can determine the variogram of a regionalized variable through a data set. However, here are some examples of the most common models of isotropic variograms, with $r = |h|$, a being the range and c being the sill :

- Spherical model :

$$\gamma(h) = \begin{cases} c\left[1.5\,\frac{r}{a} - 0.5\left(\frac{r}{a}\right)^3\right] & \text{if} \quad r \leq a \\ c & \text{if} \quad r \geq a \end{cases} \tag{1.16}$$

- Cubic model :

$$\gamma(h) = \begin{cases} c\left[7\left(\frac{r}{a}\right)^2 - 8.75\left(\frac{r}{a}\right)^3 + 3.5\left(\frac{r}{a}\right)^5 - 0.75\left(\frac{r}{a}\right)^7\right] & \text{if} \quad r \leq a \\ c & \text{if} \quad r \geq a \end{cases} \tag{1.17}$$

- Exponential model :

$$\gamma(h) = c\left[1 - \exp\left(-\frac{r}{a}\right)\right] \tag{1.18}$$

- Gaussian model :

$$\gamma(h) = c\left[1 - \exp\left(-\frac{r^2}{a^2}\right)\right] \tag{1.19}$$

- Power model, with a power $0 < \alpha < 2$ and a positive slope c :

$$\gamma(h) = c\,.\,r^{\alpha} \tag{1.20}$$

The representation of these variogram functions can be seen in Figure 1.2.

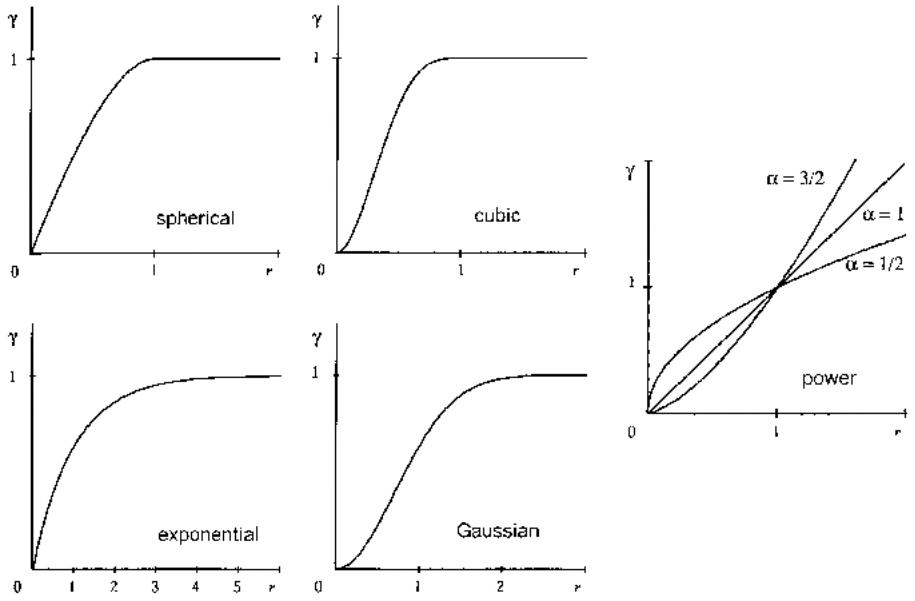


Figure 1.2: Variogram models with unit sill and scale parameters, from *Chilès and Delfiner* (1999).

1.3 Kriging Basics

Kriging is an interpolation method, thus its goal is to estimate the value of the studied regionalized variable $Z(x)$ (considered an RF) at a point x_0 where its value is unknown, using a linear combination of the N neighboring data :

$$Z^\star(x_0) = \sum_{i=1}^{N} \lambda_i(x_0)\, Z(x_i) + \lambda_0(x_0) \tag{1.21}$$

The weights λ_i associated to the regionalized variable data are chosen to make an unbiased estimate, whose variance is minimum. These weights depend on the location of the data points and their distance to the estimated point, and on the structure of the spatial dependency. In fact, kriging is the name given to the Best Linear Unbiased Estimator (BLUE).

Kriging is also the first interpolation method to take into account the spatial dependency structure of data. In fact, from a physical point of view, the RF $Z(x)$ can be described as the following decomposition :

$$Z(x) = m(x) + R(x) \tag{1.22}$$

In this equation, $m(x)$ is a smooth deterministic function that describes the systematic aspect of the phenomenon and is usually called the mean (it is indeed the mean of the RF $Z(x)$) when $m(x)$ is constant and the drift otherwise. $R(x)$ is a zero-mean RF, called the residual, whose spatial variation structure is known, and which captures the erratic fluctuations of the RF $Z(x)$.

The structure of the function $m(x)$ determines the type of kriging processed :

- Simple Kriging (SK) : $m(x) = m$ is a known constant, $Z(x)$ is supposed an SRF.
- Ordinary Kriging (OK) : $m(x) = m$ is an unknown constant, $Z(x)$ is supposed an IRF.

- Universal Kriging (UK) :

$$m(x) = \sum_{l=0}^{L} a_l \, f^l(x) \tag{1.23}$$

  In which the $f^l(x)$ functions are known basis functions and a_l are fixed but unknown coefficients.

Four constraints sum up the kriging process :

1. **Linearity constraint**
   The estimate has to be a linear combination of the data, and thus has to be written as in equation (1.21).

2. **Authorization constraint**
   The expectation and the variance of the estimate error $Z^\star(x_0) - Z(x_0)$ have to exist. This constraint is used only when the residual $R(x)$ is considered an IRF.

3. **Unbiasedness constraint**
   The kriging estimate must be unbiased. That means that $E[Z^\star(x_0) - Z(x_0)] = 0$. A direct consequence of this constraint is that kriging is an exact interpolator.

4. **Optimality constraint**
   The weights λ_i are determined in order to minimize $Var[Z^\star(x_0) - Z(x_0)]$ while following the other constraints. This makes kriging a smoothing interpolator, as that implies $Var[Z^\star(x_i)] \leq Var[Z(x_i)]$.

These constraints lead to the linear system of equations that will be solved to determine the kriging weights and find the estimate. In the next section, is explained how to solve the Universal Kriging system.

1.4 Universal Kriging or Kriging with a Trend Model

Universal Kriging doesn't require the validity of some stationarity hypothesis, as opposed to Simple or Ordinary Kriging. In particular, it takes into account any possible drift of the regionalized variable. Applied to the hydraulic head, that means Universal Kriging is able to take into account the existence of a hydraulic gradient, which is more often than not different from nil due to the flow conditions. That explains why this kriging method is used in the further developments of this thesis and why its basics are detailed below.

1.4.1 Drift Terms

As explained in section 1.3, in the Universal Kriging method, the RF $Z(x)$ can be described as in equations (1.22) and (1.23) :

$$Z(x) = m(x) + R(x) \qquad \text{with} \qquad m(x) = \sum_{l=0}^{L} a_l \, f^l(x)$$

In order to solve the kriging system, one has to determine the $f^l(x)$ functions that define the trend. Ideally, they should be specified by the physics of the problem. Though, in the absence of any information about the shape of the trend, the dichotomization of the Z data into trend and residual components is somewhat arbitrary : what is regarded as stochastic fluctuations $R(x)$ at large scale may later be modeled as a trend if additional data allow focusing on the smaller-scale variability. In the absence of a physical interpretation, the trend is usually modeled as a low-order (≤ 2) polynomial of the coordinates of x, i.e. `x` and `y` in our 2D case.

In GSLIB's algorithm, nine drift terms can be included in the kriging system on top of the constant term :

- linear terms in `x`, `y` or `z`,
- quadratic terms in $\mathtt{x}^2$, $\mathtt{y}^2$ or $\mathtt{z}^2$,
- cross quadratic terms in `xy`, `xz` or `yz`.

As we are kriging the hydraulic head in 2D, we obviously won't use the terms in `z`. So, an example of a possible trend model in our case would be a linear 2D one :

$$m(x) = \mu_0 + \mu_1\,\mathtt{x} + \mu_2\,\mathtt{y}$$

1.4.2 Linearity Constraint

As previously mentioned in equation (1.21), $Z(x)$ has to be a linear combination of the $Z(x_i)$ data and thus it is written :

$$Z^\star(x_0) = \sum_{i=1}^{N} \lambda_i(x_0)\, Z(x_i) + \lambda_0(x_0)$$

1.4.3 Authorization Constraint

In the Universal Kriging method, the residual function $R(x)$ only follows the intrinsic hypothesis. That means only linear combinations of increments $\big(R(x+h)-R(x)\big)$ have second order moments necessarily defined. Then the estimate error has to be a linear combination of increments of the residual function $R(x)$ to be sure it has a variance.

The following can be written for the estimate error :

$$\begin{aligned}
Z^\star(x_0) - Z(x_0) &= \lambda_0(x_0) + \sum_{i=1}^{N} \lambda_i(x_0)\, Z(x_i) - Z(x_0) \\
&= \lambda_0(x_0) + \sum_{i=1}^{N} \lambda_i(x_0)\, \big[m(x_i) + R(x_i)\big] - m(x_0) - R(x_0) \\
&= \underbrace{\lambda_0(x_0) + \sum_{i=1}^{N} \lambda_i(x_0)\, m(x_i) - m(x_0)}_{\text{non random terms}} + \sum_{i=1}^{N} \lambda_i(x_0)\, R(x_i) - R(x_0)
\end{aligned}$$

$Z^\star(x_0) - Z(x_0)$ is a linear combination of increments of the residual function $R(x)$ if and only if :

$$\lambda_0(x_0) + \sum_{i=1}^{N} \lambda_i(x_0)\, m(x_i) - m(x_0) = 0 \tag{1.24}$$

1.4.4 Unbiasedness Constraints

To have an unbiased estimate, the following condition has to be true, with the simplified notation $\lambda_i = \lambda_i(x_0)$:

$$\begin{aligned}
E[Z^\star(x_0) - Z(x_0)] &= 0 \\
E\left[\lambda_0 + \sum_{i=1}^{N} \lambda_i\, Z(x_i) - Z(x_0)\right] &= 0 \\
\lambda_0 + \sum_{i=1}^{N} \lambda_i\, m(x_i) - m(x_0) &= 0 \\
\lambda_0 + \sum_{i=1}^{N} \lambda_i \sum_{l=0}^{L} a_l\, f^l(x_i) - \sum_{l=0}^{L} a_l\, f^l(x_0) &= 0 \\
\lambda_0 + \sum_{l=0}^{L} a_l \left[\sum_{i=1}^{N} \lambda_i\, f^l(x_i) - f^l(x_0)\right] &= 0
\end{aligned}$$

This is true if :

$$\lambda_0 = 0 \qquad \text{and} \qquad \forall l = 0, \ldots, L, \quad \sum_{i=1}^{N} \lambda_i\, f^l(x_i) - f^l(x_0) = 0 \tag{1.25}$$

One can see that, under such conditions, the above-said authorization constraint is *de facto* met. Unbiasedness and authorization constraints actually coincide. Since $f^0(x)$ has to be set equal to 1, $\forall x$, since the mean is unknown, for $l = 0$, $\sum_{i=1}^{N} \lambda_i\, f^l(x_i) = f^l(x_0)$ becomes :

$$\sum_{i=1}^{N} \lambda_i = 1 \tag{1.26}$$

So we have to work with a sum of kriging weights equal to 1.

The estimate then becomes :

$$Z^\star(x_0) = \sum_{i=1}^{N} \lambda_i\, Z(x_i) \quad \text{with} \quad \sum_{i=1}^{N} \lambda_i\, f^l(x_i) = f^l(x_0) \quad \forall l = 0, \ldots, L$$

1.4.5 Optimality Constraint

The optimality constraint goal is to minimize the estimation variance. Using the unbiasedness constraints (1.25) and the result (1.14), and assuming that the covariance function C is defined for the RF Z[1] :

$$
\begin{aligned}
Var[Z^\star(x_0) - Z(x_0)] \quad &= \\
&= Var\left[\sum_{i=1}^{N} \lambda_i\, Z(x_i) - Z(x_0)\right] \\
&= Var\left[\sum_{i=1}^{N} \lambda_i \Big(Z(x_i) - Z(x_0)\Big)\right] \\
&= E\left[\left[\sum_{i=1}^{N} \lambda_i \Big(Z(x_i) - Z(x_0)\Big)\right]^2\right] \quad \text{since} \quad E\left[\sum_{i=1}^{N} \lambda_i \Big(Z(x_i) - Z(x_0)\Big)\right] = 0 \\
&= E\left[\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\,\lambda_j \Big(Z(x_i) - Z(x_0)\Big)\Big(Z(x_j) - Z(x_0)\Big)\right] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\,\lambda_j\, E\Big[\big(Z(x_i) - Z(x_0)\big)\big(Z(x_j) - Z(x_0)\big)\Big] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\,\lambda_j \Big[C(x_i - x_j) - C(x_i - x_0) - C(x_j - x_0) + C(0)\Big] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\,\lambda_j\, C\left(x_i - x_j\right) - 2\sum_{i=1}^{N} \lambda_i\, C\left(x_i - x_0\right) + C(0)
\end{aligned}
$$

Under the unbiasedness constraint, the problem is now to find N weights λ_i minimizing $Var[Z^\star(x_0) - Z(x_0)]$. This is classically solved by the method of Lagrange multipliers. We consider the function :

$$
Q = Var[Z^\star(x_0) - Z(x_0)] + 2\sum_{l=0}^{L} \mu_l \left[\sum_{i=1}^{N} \lambda_i\, f^l(x_i) - f^l(x_0)\right]
$$

where $2\,\mu_l = 2\,\mu_l(x_0)$, $l = 0, \ldots, L$, are $L+1$ additional unknowns, the Lagrange multipliers, and determine the unconstrained minimum of Q by equating the partial derivatives of Q to zero.

$$
\begin{aligned}
\frac{\partial Q}{\partial \lambda_i} &= 2\sum_{j=1}^{N} \lambda_j\, C(x_i - x_j) + 2\sum_{l=0}^{L} \mu_l\, f^l(x_i) - 2\, C(x_i - x_0) = 0 \qquad && \forall i = 1, \ldots, N \\
\frac{\partial Q}{\partial \mu_l} &= 2\left[\sum_{i=1}^{N} \lambda_i\, f^l(x_i) - f^l(x_0)\right] = 0 && \forall l = 0, 1, \ldots, L
\end{aligned}
$$

The fact the extremum is indeed a minimum is guaranteed by the convexity of $Var[Z^\star(x_0) - Z(x_0)]$ as a function of the λ_i. This leads to the following set of $N + L + 1$ linear equations

[1] This is not a prerequisite in Universal Kriging, but GSLIB works with covariance models, that is why we use them instead of the variograms here.

with $N + L + 1$ unknowns :

$$\begin{cases} \sum_{j=1}^{N} \lambda_j \, C(x_i - x_j) + \sum_{l=0}^{L} \mu_l \, f^l(x_i) = C(x_i - x_0) & \forall i = 1, \ldots, N \\ \sum_{i=1}^{N} \lambda_i \, f^l(x_i) = f^l(x_0) & \forall l = 0, \ldots, L \end{cases} \tag{1.27}$$

In matrix notations, the Universal Kriging system (1.27) is of the form $\mathbf{AX}=\mathbf{B}$ with the following structure :

$$\underbrace{\begin{pmatrix} C_{ij} & f_i^l \\ f_j^l & 0 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} \lambda_j \\ \mu_l \end{pmatrix}}_{\mathbf{X}} = \underbrace{\begin{pmatrix} C_{i0} \\ f_0^l \end{pmatrix}}_{\mathbf{B}}$$

With the simplified notations detailed below :

$$C_{ij} = \begin{pmatrix} C(x_1 - x_1) & \ldots & C(x_1 - x_N) \\ \vdots & & \vdots \\ C(x_N - x_1) & \ldots & C(x_N - x_N) \end{pmatrix} \qquad f_i^l = \begin{pmatrix} 1 & f^1(x_1) & \ldots & f^1(x_N) \\ \vdots & \vdots & & \vdots \\ 1 & f^L(x_1) & \ldots & f^L(x_N) \end{pmatrix} \qquad f_j^l = {}^t(f_i^l)$$

$$\lambda_j = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_N \end{pmatrix} \qquad \mu_l = \begin{pmatrix} \mu_0 \\ \vdots \\ \mu_L \end{pmatrix} \qquad C_{i0} = \begin{pmatrix} C(x_1 - x_0) \\ \vdots \\ C(x_N - x_0) \end{pmatrix} \qquad f_0^l = \begin{pmatrix} 1 \\ f^1(x_0) \\ \vdots \\ f^L(x_0) \end{pmatrix}$$

The kriging variance is obtained by premultiplying the first N equations of (1.27) by λ_i, summing over i, and then using the last $(L+1)$ equations. The result is the Universal Kriging variance :

$$\sigma^2_{UK} = E[Z^\star(x_0) - Z(x_0)]^2 = C(0) - \sum_{i=1}^{N} \lambda_i \, C(x_i - x_0) - \sum_{l=0}^{L} \mu_l \, f^l(x_0) \tag{1.28}$$

1.4.6 Solving the Kriging Equations

1.4.6.1 Conditions for Nonsingularity

The linear system (1.27) has a unique solution if and only if its matrix $\mathbf{A}$ is nonsingular. This holds under the following set of sufficient conditions :

1. That the submatrix (C_{ij}) is strictly positive definite,
2. That the submatrix (f_i^l) is of full rank $L + 1$ (equal to the number of columns).

The proof follows from straightforward matrix algebra.

Strict positive definiteness of (C_{ij}) is ensured by the use of a strictly positive definite covariance function and the elimination of duplicate data points. The condition on (f_i^l)

expresses that the $L+1$ basis functions $f^l(x)$ are linearly independent on the spatial distribution S :

$$\sum_{l=0}^{L} c_l \, f^l(x) \quad \forall x \in S \qquad \Rightarrow \qquad c_l = 0 \; : \quad l = 0, \ldots, L$$

This is a standard condition of "sampling design". For one thing there must be at least as many data points as there are basis functions (thus $N \geq L+1$). Moreover the arrangement of the points must provide enough constraints to allow the determination of the coefficients a_l in the linear model (1.23). A counterexample in 2D is when $m(x)$ is a plane and all sample points are aligned : obviously the plane is not constrained by a single line. Likewise, when $m(x)$ is a quadratic function, the system is singular if all data points lie along two lines, a circle, an ellipse, a parabola, or a hyperbola. In view of these remarks, one must be careful, particularly when using moving neighborhoods, not to create singular systems by a bad selection of the data points.

1.4.6.2 Computing the Solution

In GSLIB, the kriging system of linear equations (1.27) is solved by the classic Gaussian elimination algorithm, with a use of partial pivoting to take into account that the matrix **A** is not positive definite in Universal Kriging.

1.5 Multivariate Geostatistics : Cokriging

The goal of multivariate geostatistics is to improve the estimate using the correlation between several regionalized variables. In particular, this can improve the results when the studied regionalized variable is undersampled and when there is data of one or several correlated variable(s) available.

For example, the cokriging estimate for the regionalized variable z at a point x_0, with some correlated data of the regionalized variable y is a linear combination :

$$z^\star(x_0) = \lambda_0(x_0) + \sum_{i=1}^{N} \lambda_i(x_0) \, z(x_i) + \sum_{s=1}^{S} \theta_s(x_0) \, y(x_s) \tag{1.29}$$

The weights of this linear combination are chosen to minimize the estimate variance under an unbiasedness constraint, as in kriging. In order to do this, all the regionalized variables are considered as random functions, even the secondary variables. That means that the spatial dependency of all the considered variables is taken into account. The cokriging equations won't be detailed here, as cokriging applied to hydrogeology is discussed further.

Chapter 2

Kriging under Boundary Conditions

Kriging as described in Chapter 1, when used to draw contour maps of the hydraulic head, is strictly an interpolation tool. Its main advantage on other interpolation methods is its ability to take into account the spatial variability of the data. Besides this, kriging is only using the data and their location to make an estimate. This chapter describes how we can introduce in the kriging data a key component, used to solve the diffusivity equation in hydrogeology, the boundary conditions.

2.1 Boundary Conditions in Hydrogeology

As explained in *de Marsily* (1981), the diffusivity equation that dictates the flow in hydrogeology is often written as following, with the usual simplifications(cf. *de Marsily* (1981), chapter 5) :

$$\text{div } (T \,\text{grad}\, h) = S \frac{\partial h}{\partial t} + Q \tag{2.1}$$

with :

- T, the tensor of order 2 of transmissivity $[L^2.T^{-1}]$,
- h, the hydraulic head $[L]$,
- S, the storage coefficient $[.]$,
- Q, the total discharge $[L^2.T^{-1}]$.

In the following, we will assume a 2D steady state flow ($\frac{\partial h}{\partial t} = 0$), that Q is nil and that T is isotropic and, for the time being, constant. A simplified equation can thus be derived from (2.1) :

$$\nabla^2 h = \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = \frac{Q}{T} = 0 \tag{2.2}$$

In order to solve this partial differential equation for a concrete case, one needs to determine the boundary conditions on the integration domain. There are three usual types of boundary conditions :

- the Dirichlet conditions, on the variable itself : prescribed h,

- the Neumann conditions, on the first derivative of the variable : prescribed $\frac{\partial h}{\partial n}$,
- the Fourier conditions, on both h and $\frac{\partial h}{\partial n}$: prescribed $h + \alpha \frac{\partial h}{\partial n}$.

A fourth type of boundary condition can be added : it concerns specific double conditions (a head condition and a distinct gradient one) such as the phreatic surface or a seepage surface. But this type of boundary is only encountered in 3D flow. Therefore they won't be detailed below. The Fourier conditions would normally fall into the same category as Dirichlet and Neumann conditions, but their introduction into the kriging equations is slightly trickier. So they won't be detailed below either.

2.1.1 Prescribed Head

Dirichlet conditions are required on boundaries where the hydraulic head on the boundary doesn't depend on the flow conditions in the aquifer. It will generally be where the aquifer is in direct contact with free water, such as a river, a lake or a sea. Along this contact between the aquifer and the river(or lake, sea...), the hydraulic head is constant and prescribed by the water elevation in the river. The river can either feed or drain the aquifer. Of course, the water level in the river can change along its course, but the river still prescribes the hydraulic head along the boundary.

It is pretty obvious to imagine how we can take into account this type of boundary condition in the kriging system : by discretizing the continuous boundary into a finite number of data points which will be assigned the prescribed head value(s). We will simply add these points to the data points provided by water table measurements.

2.1.2 Prescribed Flux

This is the Neumann condition in hydrogeology. According to the Darcy law, prescribing the head gradient normal to the boundary, $\frac{\partial h}{\partial n}$, is indeed the same as prescribing the flux $-T \frac{\partial h}{\partial n}$ on this boundary, provided T is known. There are two distinct conditions of prescribed flux :

- The no flow boundaries : $\frac{\partial h}{\partial n} = 0$. For example, in a 2D flow, the contact between an aquifer and neighboring impervious formations.
- The prescribed flux with a value different than 0. For example, runoff water entering an aquifer along a boundary.

Introducing data that specifies the gradient component normal to the prescribed flux boundary is not something as obvious as adding head data points. It necessitates cokriging with head gradient data, i.e. adding the flux (or head gradient) as a secondary variable.

In principle, since the prescribed flux contour is continuous, one should consider a continuous cokriging estimator. But in the same way as for prescribed head conditions, the head gradient will only be specified at discrete points along the prescribed flux contour. The

(co-)kriging estimate would then be, with the simplified notation $\lambda_i = \lambda_i(x_0)$ and $\theta_s = \theta_s(x_0)$, and h represented as the RF Z :

$$Z^\star(x_0) = \lambda_0 + \sum_{i=1}^{N} \lambda_i \, Z(x_i) + \sum_{s=1}^{S} \theta_s \, Z'(x_s) \tag{2.3}$$

A further simplification can be introduced, replacing gradients by finite differences. In practice, it suffices to discretize the problem and replace the orthogonal gradient component by the differences between pairs of dummy points : one of the dummy points is on one side of the boundary while the other is on the other side, the two points drawing a segment perpendicular to the boundary, as depicted in Figure 2.1.

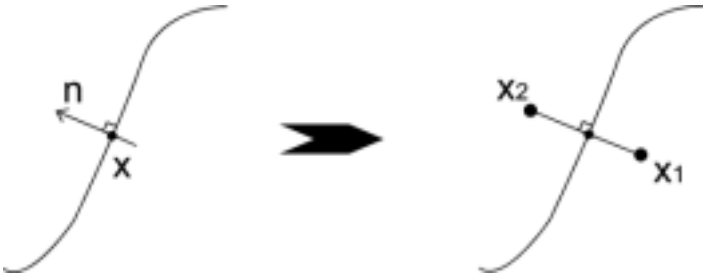


Figure 2.1: From gradient to a pair of dummy points.

Z being the variable estimated (the hydraulic head h), this figure is in fact the graphic representation of the mathematical approximation :

$$\frac{\partial Z}{\partial n}(x) \approx \frac{Z(x_1) - Z(x_2)}{|x_1 - x_2|} \tag{2.4}$$

The kriging estimate can then be written :

$$Z^\star(x_0) = \lambda_0 + \sum_{i=1}^{N} \lambda_i \, Z(x_i) + \sum_{s=1}^{S} \theta_s \, [Z(x_{s_1}) - Z(x_{s_2})] \tag{2.5}$$

In the no flow boundary case, one notices that the differences $Z(x_{s_1}) - Z(x_{s_2})$ are zeros. Why then consider these differences at all since their contribution to the estimator is nil ? Because, and that is key, the weights λ_i are different from kriging weights based on the $Z(x_i)$ alone.

This was first presented by *Delhomme* (1979) at a conference but never published. We worked together to bring this one time application into a widespread robust algorithm.

2.2 The Kriging under Boundary Conditions System

The same workflow as for Universal Kriging is applied.

2.2.1 Linearity Constraint

As previously mentioned in equation (2.5), $Z(x)$ has to be a linear combination of the $Z(x_i)$ and $Z\left(x_{s_1}\right) - Z\left(x_{s_2}\right)$ data. Thus it is written :

$$Z^\star(x_0) = \lambda_0 + \sum_{i=1}^{N} \lambda_i \, Z(x_i) + \sum_{s=1}^{S} \theta_s \, \left[Z\left(x_{s_1}\right) - Z\left(x_{s_2}\right)\right]$$

2.2.2 Authorization Constraint

Similarly to the Universal Kriging method, we can write for the estimate error :

$$\begin{aligned}
Z^\star(x_0) - Z(x_0) &= \lambda_0 + \sum_{i=1}^{N} \lambda_i \, Z(x_i) + \sum_{s=1}^{S} \theta_s \, \left[Z\left(x_{s_1}\right) - Z\left(x_{s_2}\right)\right] - Z(x_0) \\
&= \lambda_0 + \sum_{i=1}^{N} \lambda_i \left[m(x_i) + R(x_i)\right] - m(x_0) - R(x_0) \\
&\quad + \sum_{s=1}^{S} \theta_s \, \left[m\left(x_{s_1}\right) + R\left(x_{s_1}\right) - m\left(x_{s_2}\right) - R\left(x_{s_2}\right)\right] \\
&= \underbrace{\lambda_0 + \sum_{i=1}^{N} \lambda_i \, m(x_i) + \sum_{s=1}^{S} \theta_s \, \left[m\left(x_{s_1}\right) - m\left(x_{s_2}\right)\right] - m(x_0)}_{\text{non random terms}} \\
&\quad + \sum_{i=1}^{N} \lambda_i \, R(x_i) - R(x_0) + \sum_{s=1}^{S} \theta_s \, \left[R\left(x_{s_1}\right) - R\left(x_{s_2}\right)\right]
\end{aligned}$$

As in Universal Kriging, $Z^\star(x_0) - Z(x_0)$ is a linear combination of increments of the residual function $R(x)$ if and only if :

$$\lambda_0 + \sum_{i=1}^{N} \lambda_i \, m(x_i) + \sum_{s=1}^{S} \theta_s \, \left[m\left(x_{s_1}\right) - m\left(x_{s_2}\right)\right] - m(x_0) = 0 \tag{2.6}$$

2.2.3 Unbiasedness Constraints

To have an unbiased estimate, the following condition has to be true :

$$\begin{aligned}
E[Z^\star(x_0) - Z(x_0)] &= 0 \\
E\left[\lambda_0 + \sum_{i=1}^{N} \lambda_i\, Z(x_i) + \sum_{s=1}^{S} \theta_s\, \left[Z\left(x_{s_1}\right) - Z\left(x_{s_2}\right)\right] - Z(x_0)\right] &= 0 \\
\lambda_0 + \sum_{i=1}^{N} \lambda_i\, m(x_i) + \sum_{s=1}^{S} \theta_s\, \left[m\left(x_{s_1}\right) - m\left(x_{s_2}\right)\right] - m(x_0) &= 0 \\
\lambda_0 + \sum_{i=1}^{N} \lambda_i \sum_{l=0}^{L} a_l\, f^l(x_i) + \sum_{s=1}^{S} \theta_s \sum_{l=0}^{L} a_l\, \left[f^l\left(x_{s_1}\right) - f^l\left(x_{s_2}\right)\right] - \sum_{l=0}^{L} a_l\, f^l(x_0) &= 0 \\
\lambda_0 + \sum_{l=0}^{L} a_l \left[\sum_{i=1}^{N} \lambda_i\, f^l(x_i) + \sum_{s=1}^{S} \theta_s\, \left[f^l\left(x_{s_1}\right) - f^l\left(x_{s_2}\right)\right] - f^l(x_0)\right] &= 0
\end{aligned}$$

This is true if $\lambda_0 = 0$ and :

$$\forall l = 0, \ldots, L, \quad \sum_{i=1}^{N} \lambda_i\, f^l(x_i) + \sum_{s=1}^{S} \theta_s\, \left[f^l\left(x_{s_1}\right) - f^l\left(x_{s_2}\right)\right] - f^l(x_0) = 0 \tag{2.7}$$

Under such conditions, the authorization constraint is *de facto* met. Similarly to Universal Kriging, we have to set $f^0(x) = 1, \forall x$ and equation (2.7) is written for $l = 0$:

$$\sum_{i=1}^{N} \lambda_i + \sum_{s=1}^{S} \theta_s\, (1 - 1) - 1 = 0 \qquad \text{or} \qquad \sum_{i=1}^{N} \lambda_i = 1 \tag{2.8}$$

The estimate becomes :

$$\begin{aligned}
&Z^\star(x_0) = \sum_{i=1}^{N} \lambda_i\, Z(x_i) + \sum_{s=1}^{S} \theta_s\, \left[Z\left(x_{s_1}\right) - Z\left(x_{s_2}\right)\right] \\
\text{with} \quad &\sum_{i=1}^{N} \lambda_i\, f^l(x_i) + \sum_{s=1}^{S} \theta_s\, \left[f^l\left(x_{s_1}\right) - f^l\left(x_{s_2}\right)\right] = f^l(x_0) \qquad \forall l = 0, \ldots, L
\end{aligned}$$

2.2.4 Optimality Constraint

The optimality constraint minimizes the estimation variance, using the covariance of increments instead of the covariance for the prescribed flux data points. The calculation still uses the unbiasedness constraints (2.7) and the result (1.14). The derivation of the equations is not as detailed as for the Universal Kriging, as the method is the same.

$$
\begin{aligned}
Var[Z^\star(x_0) - Z(x_0)] \quad &= \\
&= Var\left[\sum_{i=1}^{N} \lambda_i\, Z(x_i) + \sum_{s=1}^{S} \theta_s \left(Z(x_{s_1}) - Z(x_{s_2})\right) - Z(x_0)\right] \\
&= Var\left[\sum_{i=1}^{N} \lambda_i \left(Z(x_i) - Z(x_0)\right) + \sum_{s=1}^{S} \theta_s \left[\left(Z(x_{s_1}) - Z(x_0)\right) - \left(Z(x_{s_2}) - Z(x_0)\right)\right]\right] \\
&= E\left[\left[\sum_{i=1}^{N} \lambda_i \left(Z(x_i) - Z(x_0)\right) + \sum_{s=1}^{S} \theta_s \left[\left(Z(x_{s_1}) - Z(x_0)\right) - \left(Z(x_{s_2}) - Z(x_0)\right)\right]\right]^2\right] \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\, \lambda_j\, E\Big[\big(Z(x_i) - Z(x_0)\big)\,\big(Z(x_j) - Z(x_0)\big)\Big] \\
&\qquad +2\sum_{i=1}^{N}\sum_{s=1}^{S} \lambda_i\, \theta_s \Big(E\Big[\big(Z(x_i) - Z(x_0)\big)\,\big(Z(x_{s_1}) - Z(x_0)\big)\Big] \\
&\qquad\qquad -E\Big[\big(Z(x_i) - Z(x_0)\big)\,\big(Z(x_{s_2}) - Z(x_0)\big)\Big]\Big) \\
&\qquad +\sum_{s=1}^{S}\sum_{t=1}^{S} \theta_s\, \theta_t \Big(E\Big[\big(Z(x_{s_1}) - Z(x_0)\big)\,\big(Z(x_{t_1}) - Z(x_0)\big)\Big] \\
&\qquad\qquad -E\Big[\big(Z(x_{s_1}) - Z(x_0)\big)\,\big(Z(x_{t_2}) - Z(x_0)\big)\Big] \\
&\qquad\qquad -E\Big[\big(Z(x_{s_2}) - Z(x_0)\big)\,\big(Z(x_{t_1}) - Z(x_0)\big)\Big] \\
&\qquad\qquad +E\Big[\big(Z(x_{s_2}) - Z(x_0)\big)\,\big(Z(x_{t_2}) - Z(x_0)\big)\Big]\Big) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\, \lambda_j\, C\left(x_i - x_j\right) + 2\sum_{i=1}^{N}\sum_{s=1}^{S} \lambda_i\, \theta_s \left[C\left(x_i - x_{s_1}\right) - C\left(x_i - x_{s_2}\right)\right] \\
&\qquad +\sum_{s=1}^{S}\sum_{t=1}^{S} \theta_s\, \theta_t \left[C\left(x_{s_1} - x_{t_1}\right) - C\left(x_{s_1} - x_{t_2}\right) - C\left(x_{s_2} - x_{t_1}\right) + C\left(x_{s_2} - x_{t_2}\right)\right] \\
&\qquad -2\sum_{i=1}^{N} \lambda_i\, C\left(x_i - x_0\right) - 2\sum_{s=1}^{S} \theta_s \left[C\left(x_{s_1} - x_0\right) - C\left(x_{s_2} - x_0\right)\right] + C(0)
\end{aligned}
$$

We now have to find the N weights λ_i and S weights θ_s minimizing $Var[Z^\star(x_0) - Z(x_0)]$, still with the method of Lagrange multipliers. Thus we consider the same function :

$$
Q = Var[Z^\star(x_0) - Z(x_0)] + 2\sum_{l=0}^{L} \mu_l \left[\sum_{i=1}^{N} \lambda_i\, f^l(x_i) - f^l(x_0)\right]
$$

in which $2\,\mu_l = 2\,\mu_l(x_0)$, $l = 0, \ldots, L$, are $L{+}1$ additional unknowns, the Lagrange multipliers, and determine the unconstrained minimum of Q by equating the partial derivatives of Q to

zero. The simplified notations $C_{ij} = C(x_i - x_j)$ and $f_i^l = f^l(x_i)$ are used, $\forall i, j$.

$$\begin{aligned}
\frac{\partial Q}{\partial \lambda_i} &= 2\sum_{j=1}^{N} \lambda_j\, C_{ij} + 2\sum_{s=1}^{S} \theta_s \left[C_{is_1} - C_{is_2}\right] + 2\sum_{l=0}^{L} \mu_l\, f_i^l - 2\, C_{i0}) = 0 && \forall i = 1, \ldots, N \\
\frac{\partial Q}{\partial \theta_s} &= 2\sum_{i=1}^{N} \lambda_i \left[C_{is_1} - C_{is_2}\right] + 2\sum_{t=1}^{S} \theta_t \left[C_{s_1t_1} - C_{s_1t_2} - C_{s_2t_1} + C_{s_2t_2}\right] \\
&\quad + 2\sum_{l=0}^{L} \mu_l \left[f_{s_1}^l - f_{s_2}^l\right] - 2\left[C_{s_10} - C_{s_20}\right] = 0 && \forall s = 1, \ldots, S \\
\frac{\partial Q}{\partial \mu_l} &= 2\sum_{i=1}^{N} \lambda_i\, f_i^l + 2\sum_{s=1}^{S} \theta_s \left[f_{s_1}^l - f_{s_2}^l\right] - 2\, f_0^l = 0 && \forall l = 0, 1, \ldots, L
\end{aligned}$$

This leads to the following set of $N+S+L+1$ linear equations with $N+S+L+1$ unknowns :

$$\left\{\begin{array}{ll}
\displaystyle\sum_{j=1}^{N} \lambda_j\, C_{ij} + \sum_{s=1}^{S} \theta_s \left[C_{is_1} - C_{is_2}\right] + \sum_{l=0}^{L} \mu_l\, f_i^l = C_{i0} & \forall i = 1, \ldots, N \\
\displaystyle\sum_{i=1}^{N} \lambda_i \left[C_{is_1} - C_{is_2}\right] + \sum_{t=1}^{S} \theta_t \left[C_{s_1t_1} - C_{s_1t_2} - C_{s_2t_1} + C_{s_2t_2}\right] & \\
\quad \displaystyle + \sum_{l=0}^{L} \mu_l \left[f_{s_1}^l - f_{s_2}^l\right] = \left[C_{s_10} - C_{s_20}\right] & \forall s = 1, \ldots, S \\
\displaystyle\sum_{i=1}^{N} \lambda_i\, f_i^l + \sum_{s=1}^{S} \theta_s \left[f_{s_1}^l - f_{s_2}^l\right] = f_0^l & \forall l = 0, \ldots, L
\end{array}\right. \tag{2.9}$$

In matrix notations, the kriging system (2.9) is of the following structure :

$$\left(\begin{array}{c|c|c}
C_{ij} & C_{it_1} - C_{it_2} & f_i^l \\
\hline
\begin{array}{c} C_{js_1} \\ - \\ C_{js_2} \end{array} & \begin{array}{c} C_{s_1t_1} - C_{s_1t_2} \\ - \\ -C_{s_2t_1} + C_{s_2t_2} \end{array} & \begin{array}{c} f_{s_1}^l \\ - \\ f_{s_2}^l \end{array} \\
\hline
f_j^l & f_{t_1}^l - f_{t_2}^l & 0
\end{array}\right)
\left(\begin{array}{c} \lambda_j \\ \hline \\ \theta_t \\ \\ \hline \mu_l \end{array}\right)
=
\left(\begin{array}{c} C_{i0} \\ \hline \\ \begin{array}{c} C_{s_10} \\ - \\ C_{s_20} \end{array} \\ \\ \hline f_0^l \end{array}\right)$$

With the same simplified notations as in Universal Kriging and the ones added below. The "gradient points" have their own two-digit notation : the first one indicates the number of the "gradient point" while the second indicates the dummy point, and thus can only take the

value 1 or 2.

$$C_{js_1} - C_{js_2} = \begin{pmatrix} C(x_1 - x_{11}) - C(x_1 - x_{12}) & \dots & C(x_N - x_{11}) - C(x_N - x_{12}) \\ \vdots & & \vdots \\ C(x_1 - x_{S1}) - C(x_1 - x_{S2}) & \dots & C(x_N - x_{S1}) - C(x_N - x_{S2}) \end{pmatrix}$$

$$C_{it_1} - C_{it_2} = {}^t (C_{js_1} - C_{js_2})$$

$$C_{s_1t_1} - C_{s_1t_2} - C_{s_2t_1} + C_{s_2t_2} = \begin{pmatrix} C(x_{11} - x_{11}) - C(x_{11} - x_{12}) & \dots & C(x_{S1} - x_{11}) - C(x_{S1} - x_{12}) \\ -C(x_{12} - x_{11}) + C(x_{12} - x_{12}) & \dots & -C(x_{S2} - x_{11}) + C(x_{S2} - x_{12}) \\ \vdots & & \vdots \\ C(x_{11} - x_{S1}) - C(x_{11} - x_{S2}) & \dots & C(x_{S1} - x_{S1}) - C(x_{S1} - x_{S2}) \\ -C(x_{12} - x_{S1}) + C(x_{12} - x_{S2}) & \dots & -C(x_{S2} - x_{S1}) + C(x_{S2} - x_{S2}) \end{pmatrix}$$

$$f^l_{s_1} - f^l_{s_2} = \begin{pmatrix} f^1_{11} - f^1_{12} & \dots & f^1_{S1} - f^1_{S2} \\ \vdots & & \vdots \\ f^L_{11} - f^L_{12} & \dots & f^L_{S1} - f^L_{S2} \end{pmatrix} \qquad f^l_{t_1} - f^l_{t_2} = {}^t(f^l_{s_1} - f^l_{s_2})$$

$$\theta_t = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_S \end{pmatrix} \qquad C_{s_10} - C_{s_20} = \begin{pmatrix} C(x_{11} - x_0) - C(x_{12} - x_0) \\ \vdots \\ C(x_{S1} - x_0) - C(x_{S2} - x_0) \end{pmatrix}$$

The Kriging under Boundary Conditions variance is :

$$\begin{aligned} \sigma^2_{KBC} &= E[Z^\star(x_0) - Z(x_0)]^2 \\ &= C(0) - \sum_{i=1}^{N} \lambda_i \, C(x_i - x_0) - \sum_{s=1}^{S} \theta_s \left[C(x_{s_1} - x_0) - C(x_{s_2} - x_0) \right] - \sum_{l=0}^{L} \mu_l \, f^l(x_0) \end{aligned} \tag{2.10}$$

Finally, it has to be noticed that the matrix becomes singular in the following case :

$$\exists \; i, j \text{ and } s, \qquad Z(x_i) = Z(x_{s_1}) \text{ and } Z(x_j) = Z(x_{s_2}) \tag{2.11}$$

As a result, we have to be careful that the dummy points don't overlap some data points.

2.3 Code Implementation

To implement this new type of kriging, I used the GSLIB open source code, developed in Stanford University, and documented in *Deutsch and Journel* (1998). This code is popular both among researchers and professionals, and it is particularly used in Waterloo Hydrogeologic softwares. The code implementation can be divided in several units. The algorithms won't be detailed in this section, but they are provided in the appendixes. This section starts with a common kriging issue before presenting the main algorithms introduced.

2.3.1 Kriging Neighborhood

The kriging theory is always derived as if all the N data points were used in the estimation ; this is the so-called global neighborhood case. In practice, N may be too large to allow

computation and a "moving neighborhood" has to be used, including only a subset of the data for the estimation of each grid node (see figure 2.2). Formally, this does not change anything for a grid node taken in isolation : the content of the sampled set of points is just different. However, it may alter the relationships between estimates at different grid nodes and introduce spurious discontinuities.

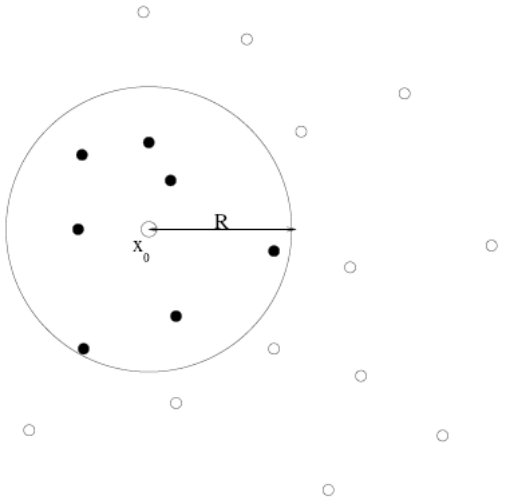


Figure 2.2: Example of selecting a subset of points with a search radius R.

In the case of the GSLIB algorithm, the kriging system not only uses the moving neighborhood method but also limits the number of points allowed in the kriging system. This was done to decrease the computational cost (this algorithm was written in the 80's and released in 1992, when computers were a lot less efficient) while selecting the closest points for each estimation. Though, the combination of these two limits proved to introduce discontinuities in the contour maps, or even to create singular matrices if the number of prescribed flux points was too important. Worse, it can strongly decrease the influence of the boundary conditions by a significant margin. To get rid of this issue, the algorithm has been modified to allow global neighborhood. All the examples presented in section 2.4 have been thus made in the global neighborhood case.

2.3.2 Adding the Boundary Conditions Data

There are two input files in GSLIB : the first one describes all the parameters the main algorithm needs and the second one provides the coordinates of the data points and the values of the studied regionalized variable, the hydraulic head in our case. The method chosen to introduce the boundary conditions data is to discretize the boundary lines into points. The user will input the coordinates and the head value for each point that is a boundary segment end. However, the program has to know if a point is a boundary segment end or not. And in the former case, it also has to know if it is a prescribed head or a prescribed flux condition. In order to solve this issue, a new input parameter has been created, named "kod". This kod is set :

- to 0 for the measured data points,

- to a positive integer value for a prescribed head segment end,
- to a negative integer value for a prescribed flux segment end.

Points that are both ends of the same segment have the same kod. More generally, if a boundary is represented by a broken line, a point will be placed at each direction shift, and all these "shift" points should have the same kod, as they are part of the same boundary. Otherwise, two points representing two different boundary lines should have a different kod.

With these added points, we now have all the needed data for the Kriging under Boundary Conditions process, but we still don't have *really* our discretized boundaries. In order to provide these, a simple algorithm will read the input data, detect the boundary points (the ones whose kod is not 0) and create several points between the two ends of the segment by linearly interpolating the coordinates (and the head value for a prescribed head segment). We have to notice here that, for this subroutine to work properly, the two ends of a boundary segment, or two consecutive shift points of a broken line, have to follow one another in the list of input data points. The data set now consists of both the real data points and the points representing the discretized boundary lines.

Finally, another algorithm scans the data set again, selects the prescribed flux points, and adds the "dummy points" in order for them to make a segment perpendicular to the boundary line they are representing. For a defined boundary, this is done using the previous and the following boundary points to compute the local slope of the boundary and setting the slope of the dummy points segment to make it perpendicular to the boundary (cf. Fig. 2.1). For the segment ends, the local slope is computed using the end point itself and its closest neighbor in the segment.

To be honest, this subroutine doesn't really add dummy points. Otherwise we would have to delete the point used to create its dummies and be careful that both dummy points are always selected when choosing the data points included in the kriging neighborhood. It's easier to just add two parameters ddx and ddy for each point of the data array. ddx and ddy will be set to 0 for each non prescribed flux data point, and to verify for a point A and its dummies A_1 and A_2 :

$$\begin{aligned} x(A_1) &= x(A) - ddx(A) & \qquad x(A_2) &= x(A) + ddx(A) \\ y(A_1) &= y(A) - ddy(A) & \qquad y(A_2) &= y(A) + ddy(A) \end{aligned}$$

Dummy points are actually computed "on the fly" when filling the kriging matrices.

2.3.3 Discretization Parameters

The two algorithms mentioned in the previous subsection both require a key parameter that has still not been defined : the spacing between two points of a discretized boundary line for one, and the spacing between the dummy points for the other. Nothing in the Kriging under Boundary Conditions theory indicates which values these parameters should be set to. However, we can easily deduce the following conditions for the first parameter :

- The more boundary points, the more precisely the boundaries will be defined and the better they will be honored.

- The more boundary points, the less real data points relatively taken into account. This is a problem if our measured data can be trusted more than the position of our boundaries.
- The more boundary points, the more computing time !

In fact, the number of data points taken into account in a kriging neighborhood is bounded (by a user defined value though) to limit the computing time. And we need $L+1$ (L being the number of drift terms) real head values to solve the kriging system. This issue was addressed by allowing kriging with a global neighborhood, but it shows that we can't add as many boundary points as we want if we choose to use a moving neighborhood. Thus, we need to find a good compromise to have enough boundary points to reproduce faithfully the boundary conditions, while still having sufficient real data points to solve the kriging system. The following solution has finally been chosen :

- if $dist \leq 2\,csiz$, $spacin = \dfrac{dist}{2}$,
- if $dist \geq 20\,csiz$, $spacin = \dfrac{dist}{20}$,
- else $spacin = csiz$.

With *dist* being the boudary segment length, *csiz* the distance between 2 consecutive nodes of tke kriging grid, and *spacin* the distance between two consecutive points of a discretized boundary segment. So we have chosen to base our discretization on the kriging grid, with a lower limit of 3 points and an upper limit of 21 points to represent a discretized segment.

The spacing between two dummy points is even harder to set. One could imagine that it represents the extension of the prescribed flux influence as, the longer the distance between the dummy points, the further the boundary condition is honored. However, practice can hardly check if this assumption is true, as no major difference has been detected when changing this parameter. It has finally been set to $2\,ciz$, thus it is also based on the kriging grid.

2.3.4 Singularity Conditions

It was already mentioned that we have to avoid dummy points overlapping some data points (cf. equation (2.11)). More precisely, the matrix becomes singular if both points of a dummy couple overlap head data points. A subroutine that scans the data array to check this has been created. If such a case occurs, we have chosen to remove the dummy points couple to protect the measured data.

2.3.5 Constant Flux Conditions

2.3.5.1 Expression of the Constant Flux

The variable data input for prescribed flux boundary points is not the hydraulic head of the point ; it is the difference between the hydraulic heads of the dummy points. When there

is no flow, the input value is pretty obvious, we have to set it to 0 : no difference in the hydraulic head between two points implies no flow between these points ! However, when the prescribed flux is a non nil constant, the input data is indeed the difference Δh between the hydraulic heads of the dummy points, i.e. a length value $[L]$. And yet we usually input a flux as a volumic flow rate $[L^3.T^{-1}]$. To provide a simple tool that any hydrogeologist can use, we have to transform this length value into a flow rate. We have from the diffusivity equation (2.2), assuming a flow along the x axis, a boundary perpendicular to the flow and subsequently the dummy points segment along the x axis too (that case will be generalized in subsubsection 2.3.5.2 below) :

$$\frac{d^2 h}{d x^2} = \frac{Q_L}{T}$$

Where Q_L is the flow rate per unit length $[L^2.T^{-1}]$ and T the transmissivity. We can twice integrate this equation between the two dummy points separated by the distance l :

$$\begin{aligned} \int_0^l \int_0^l \frac{d^2 h}{d x^2}\, dx^2 &= \int_0^l \int_0^l \frac{Q_L}{T}\, dx^2 \\ h(x=l) - h(x=0) &= \frac{Q_L}{T}\, [l-0]^2 \\ \Delta h &= \frac{Q_L\, l^2}{T} \end{aligned}$$

We can also compute the volumic flow rate Q_V by integrating the flow rate per unit length Q_L on y :

$$\begin{aligned} Q_V &= \int_0^L Q_L\, dy \\ &= Q_L\, L \end{aligned}$$

With L being the length of the boundary and e the thickness of the aquifer. We then have for Δh :

$$\Delta h = \frac{Q_V\, l^2}{T\, L} \tag{2.12}$$

So, in order to be able to input his constant flux data as a global volumic flow rate Q_V for all the boundary segment, the user also has to input a mean value for the transmissivity T. The length of the boundary L and the distance between the dummy points l are already known by the algorithm.

2.3.5.2 Constant Flux and Orthogonality

The subsubsection 2.3.5.1 above explained how to prescribe a constant flux boundary condition if the flow is orthogonal to the boundary. In fact, the segment of dummy points being perpendicular to the boundary, we will always input the flux component orthogonal to the boundary. Fortunately, that is also what hydrogeologists usually do. However, if the flow crossing the boundary is indeed not perpendicular to the boundary, there exists a colinear flux that would be nil for a flow perpendicular to the boundary. Unfortunately, the Kriging under Boundary Conditions system knows nothing about the colinear flux. So, we have to also specify this condition. This will be done by adding a second couple of dummy points, colinear to the boundary this time. And, as we have just explained, we want this colinear

flux to be nil for a flow perpendicular to the boundary. So the variable data associated to this couple of dummy points will in such a case be set to 0, as for any other no flow boundary point. However, constant flux boundary lines are now represented by twice as many points as they were before this correction.

2.3.5.3 Flux Sign

Finally, there is one issue left : the sign of the constant flux. As we can put some constant flux boundaries *inside* the study area, it was not possible to set the sign as usual, i.e. positive for an outflow, and negative for an inflow. Instead, the sign of the flow will be linked to the x and y axes :

1. If the boundary isn't colinear to the x axis :
   - A positive sign will induce a flow going towards the increasing x.
   - A negative sign will induce a flow going towards the decreasing x.
2. If the boundary is colinear to the x axis :
   - A positive sign will induce a flow going towards the increasing y.
   - A negative sign will induce a flow going towards the decreasing y.

That last issue on constant flux conditions also brings one final comment : these conditions cannot be used to represent a well. They do not allow water to be put in or out the aquifer except along external boundaries. Within the aquifer, they can only force an hydraulic head difference, which can be used to represent a known local gradient trend, whose origin can be a local change of the transmissivity for example.

2.3.6 Cubic Variogram

The GSLIB algorithm allows to choose a variogram model between all the usual models described in section 1.2.3, but one : the cubic variogram. This variogram is defined as following :

$$\gamma(h) = \begin{cases} c\left[7\left(\frac{r}{a}\right)^2 - 8.75\left(\frac{r}{a}\right)^3 + 3.5\left(\frac{r}{a}\right)^5 - 0.75\left(\frac{r}{a}\right)^7\right] & \text{if} \quad r \leq a \\ c & \text{if} \quad r \geq a \end{cases}$$

The cubic variogram is traditionally used for differentiable variables because of its nil derivative at the origin. That explains why it is commonly used for the hydraulic head variogram, its steeper slope than the Gaussian variogram also representing better the head spatial variability. Consequently, it has been added to the variogram choice in the GSLIB program.

2.3.7 Filling the Kriging under Boundary Conditions Matrix

To conclude this section, I have to mention that the main task of implementing the Kriging under Boundary Conditions process in the GSLIB algorithm was arguably to properly fill the new kriging matrix. There is no special difficulty in this task but to fully understand the structure of the GSLIB subroutine. The final algorithm is provided in Appendix A.

2.4 Application of Kriging under Boundary Conditions

This section illustrates the different improvements in the contour maps provided by the Kriging under Boundary Conditions method. The examples are pictured by a square map. By convention, as these examples are fictive, we will assume that the top of the map is the north. The graphical user interface (GUI) to implement Kriging under Boundary Conditions in *GW Contour* wasn't done yet when this thesis was written. So the contour maps have been produced with *Surfer*, using the output kriged grid and the Nearest Neighbor interpolation method to compute the kriging map from the kriged grid.

2.4.1 Comparison between Universal Kriging and Kriging under Boundary Conditions

The example detailed below consists of a study area of $500\,m \times 500\,m$, whose boundary conditions are :

- Prescribed head $h = 0\,m$ on the southern border,
- Prescribed head $h = 50\,m$ on the northwestern corner,
- No flow boundary on both the eastern and northern border.

This system has been modeled with *Visual Modflow 4.1*[1]. The output hydraulic head map is presented in figure (2.3). 12 data points have been selected on this modeled map. These

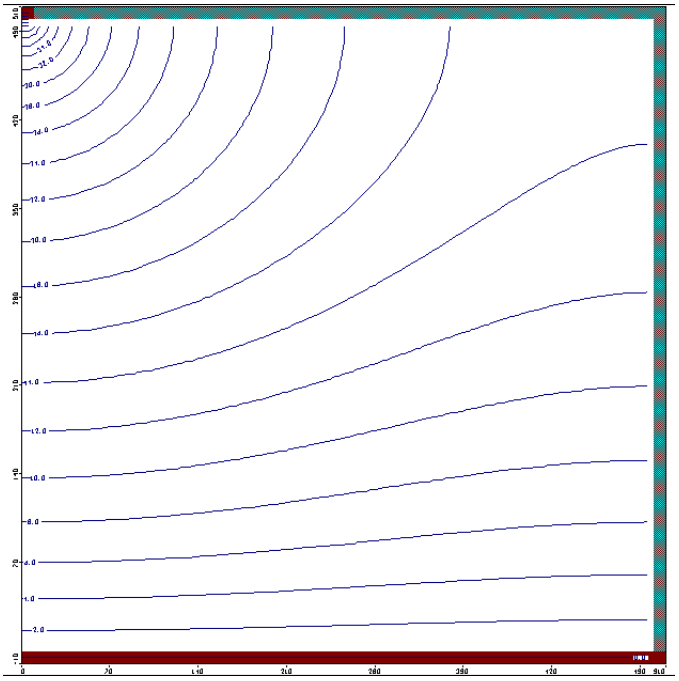


Figure 2.3: Visual Modflow modeled map. First example.

12 head values and the prescribed head in the top left corner will be the basis of our kriging example. They are identified in figure (2.4).

[1]More information available on http://www.waterloohydrogeologic.com/software/visual_modflow/visual_modflow_ov.htm

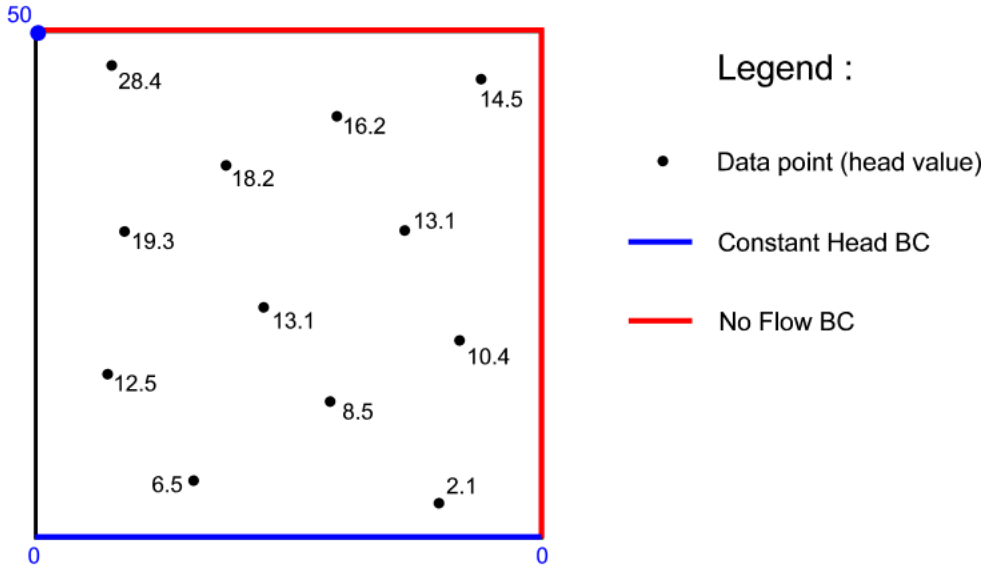


Figure 2.4: Diagram outlining the boundary conditions and the 12 data points selected. First example.

First, Universal Kriging has been applied to the set of 13 points (the northwestern prescribed head $h = 50\,m$ has been added as it is not strictly a "boundary"). The following parameters have been applied :

- Distance between 2 grid nodes $= 10\,m$,
- Constant neighborhood,
- Linear drift in x and y only considered,
- Cubic variogram, with a sill of 1, a range of $710\,m$ and a nugget effect of 0.01.

Figure 2.5 shows that the boundary conditions are not honored with the Universal Kriging, be it the prescribed head or the no flow limits.

A first step is to introduce the southern prescribed head boundary. The output result is presented in figure 2.6. It shows that the head values are indeed set to 0 on the southern border of the area. It already improves the map when we compare this one with both the Universal Kriging and the *Visual Modflow* ones.

The next step is to introduce the full boundary conditions, i.e. adding the no flow limits. The new contour map is shown in figure 2.7. It is almost identical to the Modflow model map (figure 2.3), unlike the Universal Kriging map (figure 2.5). The improvement is really noteworthy. Still, one could argue that the contour lines are not exactly perpendicular to the no flow limits, especially in the northeastern zone. This is explained by the fact that there is a boundary point in the top right corner, a point whose dummy segment slope is defined by its two neighbors. However, it happens that one of these neighbors belongs to the eastern boundary while the other belongs to the northern boundary. Obviously, the dummy segment won't be perpendicular to any of the lines, but will be the bisector of the angle between the two boundary segments. The resulting condition can explain why the contour lines are less perpendicular in this zone.

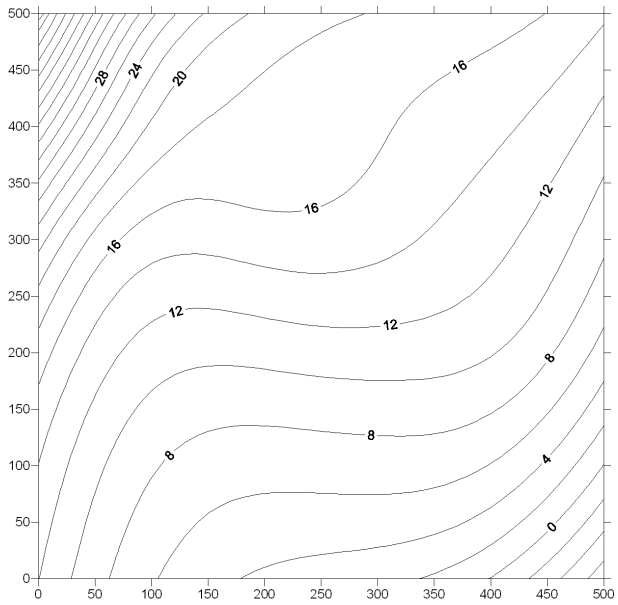


Figure 2.5: Universal Kriging contour map.

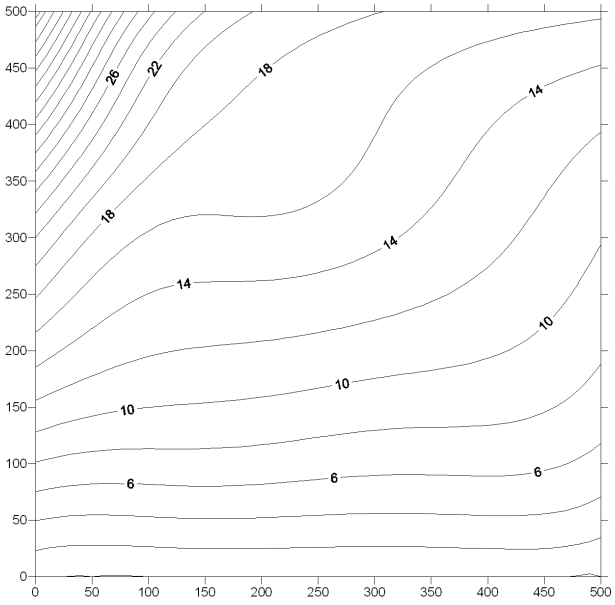


Figure 2.6: Kriging with the prescribed head condition map.

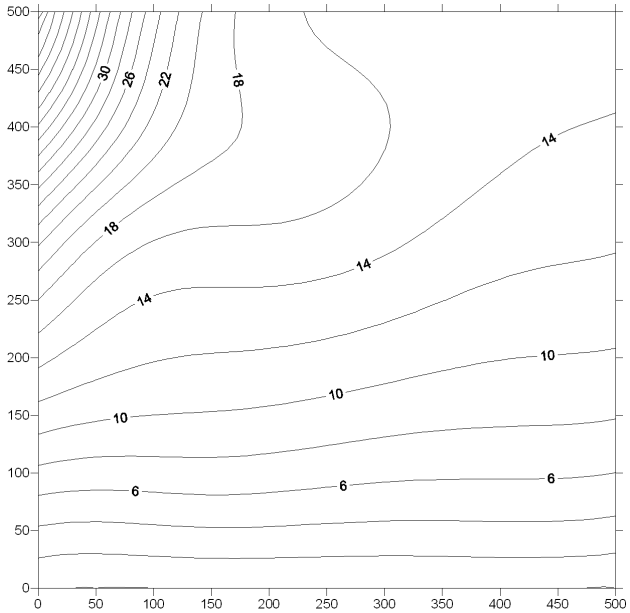


Figure 2.7: Kriging under Boundary Conditions map. First example.

2.4.2 Undefined boundaries

Another feature that has to be mentioned is that Kriging under Boundary Conditions doesn't require all the boundaries of the study area to be defined. Both Finite Difference and Finite Element Modeling programs assume that all the undefined boundaries are no flow limits. Kriging under Boundary Conditions provides a "degree of freedom" here, compared to the discretized solving of the partial differential equation. The following example illustrates this.

- Figure 2.8 presents another set of boundary conditions applied to the same study area and the new set of hydraulic head values modeled with *Visual Modflow*.
- Figure 2.9 presents the output map computed with *Visual Modflow*.
- Figure 2.10 presents the Kriging under Boundary Conditions map.

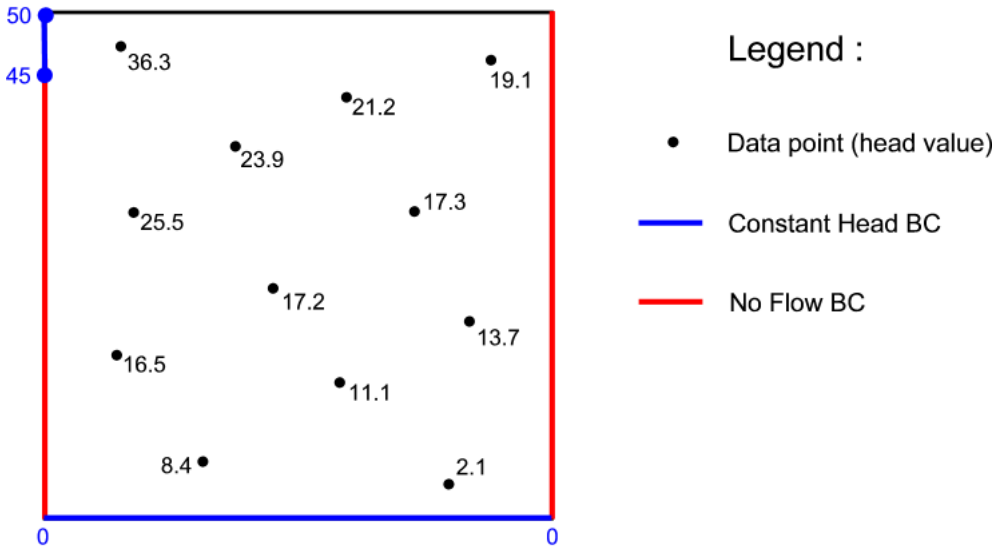


Figure 2.8: Diagram outlining the boundary conditions and the 12 data points selected. Second example.

The difference between the two maps is obvious in the northern zone : while *Visual Modflow* makes the contour lines perpendicular to the upper limit, Kriging under Boundary Conditions doesn't. The only boundary conditions honored are the ones defined by the user. This can be useful when the area to be studied has an arbitrary limit (e.g. a country border, here, the norhern limit), for which no hydrogeological conditions could be set.

2.4.3 Constant Flux Boundaries

The goal of this subsection is to highlight the issue presented in subsection 2.3.5.2, i.e. the fact that the kriging system doesn't know anything about the colinear component of the flux, if not directly specified. To illustrate this, the same example as described in figure 2.4 has been used, except that the northern no flow boundary has been replaced by a constant flux boundary. The hydraulic head difference between the two dummy points has been respectively set to $\Delta h = 5\,m$ (c.f. figure 2.11, left map) and $\Delta h = -5\,m$ (c.f. figure 2.11,

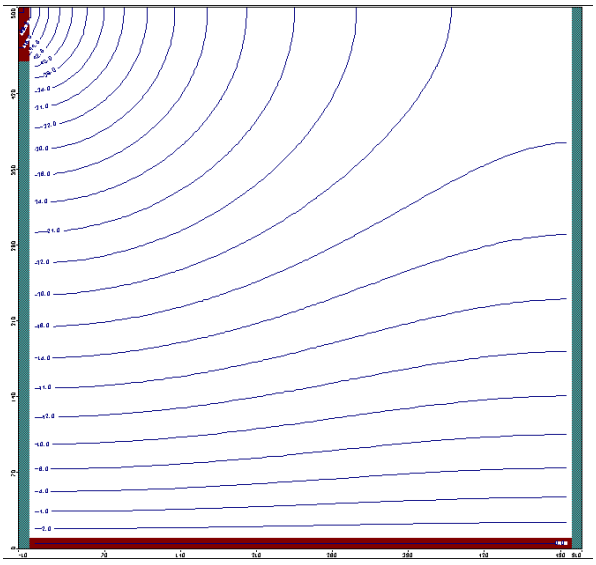


Figure 2.9: Visual Modflow modeled map. Second example.

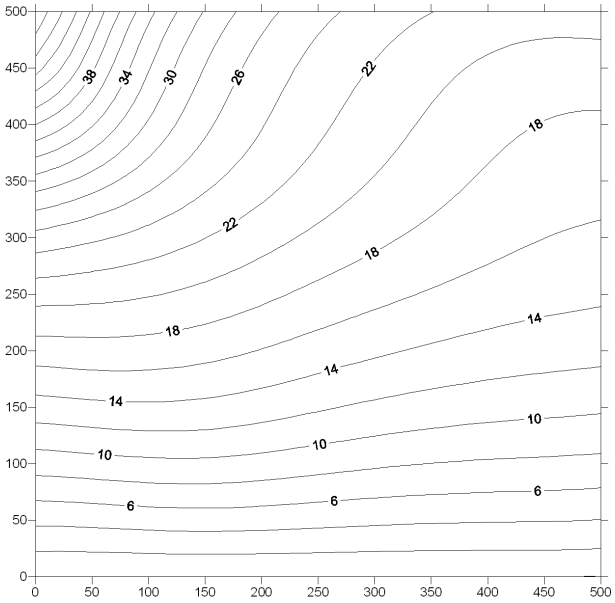
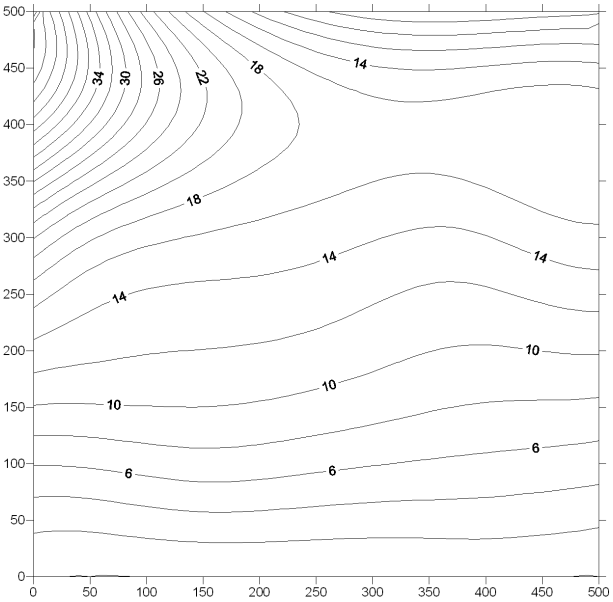


Figure 2.10: Kriging under Boundary Conditions map. Second example.



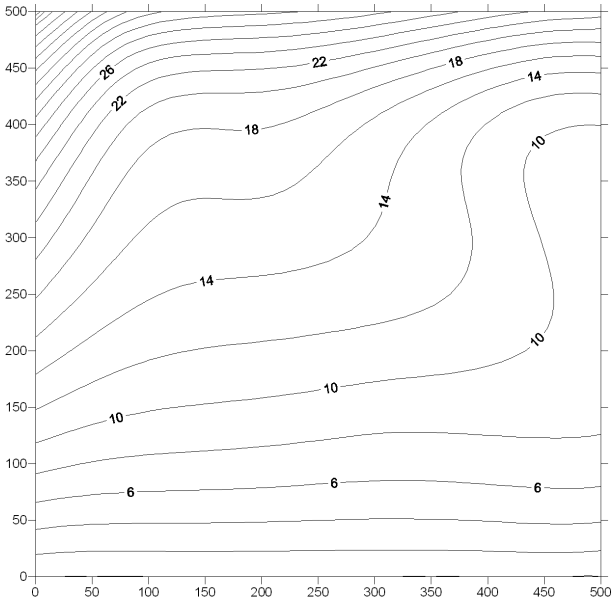


Figure 2.11: Kriging under Boundary Conditions map. Constant Flux example.

right map). The left map thus illustrates an outflow (flow going towards the increasing y, cf. 2.3.5.3), and the right map an inflow (flow going towards the decreasing y).

2.4.4 River and Inside Constant Flux

A frequently asked question during my work was to know if Kriging under Boundary Conditions allows to put some boundaries inside the study area. The answer to this question is provided in this subsection, and in subsection 2.4.5. To illustrate the river and the inside constant flux cases, a very simple example has been created. Its boundary conditions are :

- Prescribed head $h = 50\,m$ on the northern boundary, $h = 0\,m$ on the southern one,
- No flow boundary on the eastern and western limits,
- "River", i.e. prescribed head between the points $(250; 250)$ and $(250; 0)$, with a head decrasing from $h = 15\,m$ to $h = 0\,m$ or,
- Constant flux on a line between the points $(200; 200)$ and $(300; 200)$, with a local head gradient of $\Delta h = -20\,m$

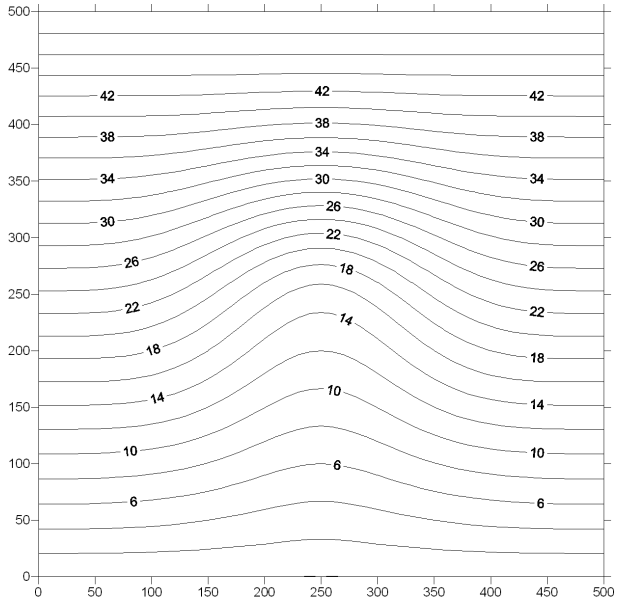


Figure 2.12: River conditioning the flow.

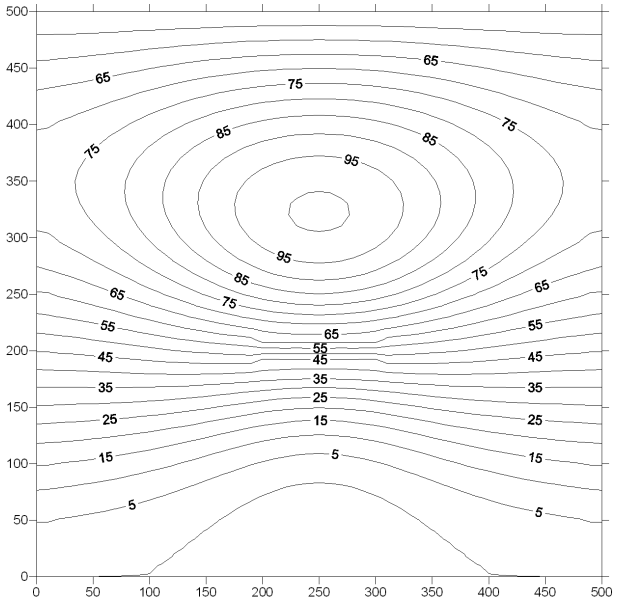


Figure 2.13: Inside constant flux condition.

The results are presented in figures 2.12 and 2.13 and are as expected :

- The hydraulic head is set to the river level along its stream, thus radically changing the shape of the whole contour map.
- The high gradient zone due to the constant flux could be interpreted as a low transmissivity zone. The head difference Δh was purposedly set to a high value, and we can see that it forces the kriging system to consider that there is a very high head hill north of the flux constraint, in order to honor both the prescribed flux and the prescribed head conditions.

2.4.5 "Screen Effect"

One case of inside boundary condition has not been mentioned yet : the no flow boundary one. To illustrate it, the same example as the one used for an inside constant flux has been created (see subsection 2.4.4). The only difference is obviously that the hydraulic head difference Δh is set to 0. The output map is presented in figure 2.14.

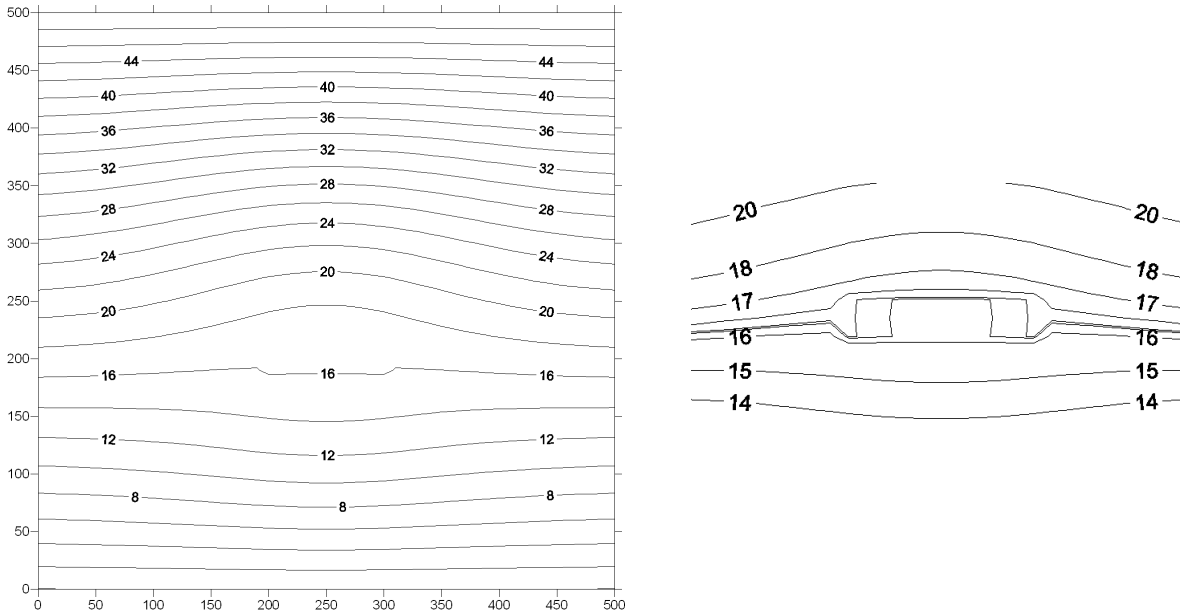


Figure 2.14: No flow boundary inside the study area.

The result is not at all what was expected and doesn't seem to follow the no flow condition as the contour lines don't look perpendicular to the inside boundary. The result surprisingly looks as if we have a higher transmissivity zone at the boundary location, as the hydraulic gradient is low there. However, further examination of this location shows that the "no flow" condition is indeed respected : if we draw the appropriate contour lines, we can see that they briefly become perpendicular to the boundary at its location. The perturbation of the hydraulic head map is minimal though.

This result makes us question the true "nature" of such a no flow condition in the middle of the study area. Thus, this example was modeled in *Visual Modflow*. The result is presented in figure 2.15. It clearly shows that the water doesn't cross the no flow boundary, and is forced to by-pass it, thus creating a discontinuity in the hydraulic head at the location of the boundary. This is what we can call the "screen effect", something that kriging cannot reflect, as it assumes that the hydraulic head is a continuous variable.

However, this issue can be solved by considering the boundary as a screen indeed, when kriging. That means that for every estimation node, the kriging neighborhood will be limited to the data points that are not "behind" the screen. An algorithm was implemented to select only the data points that are on the same side of the screen as the estimation point. Basically, it's a classic convex hull problem, and we just have to check if the segment between the data point and the estimation node crosses the screen segment or not. This is well explained by *Erickson* (2002) [2]. The kriged map with the screen effect is presented in figure 2.16.

[2] http://compgeom.cs.uiuc.edu/~jeffe/teaching/373/notes/x05-convexhull.pdf and ...x06-sweepline.pdf

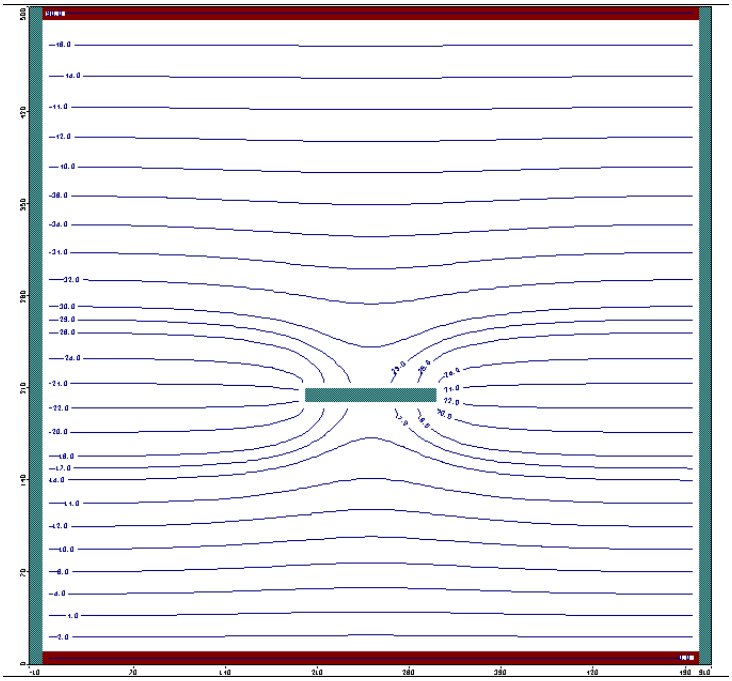


Figure 2.15: Visual Modflow modeled map. Screen effect.

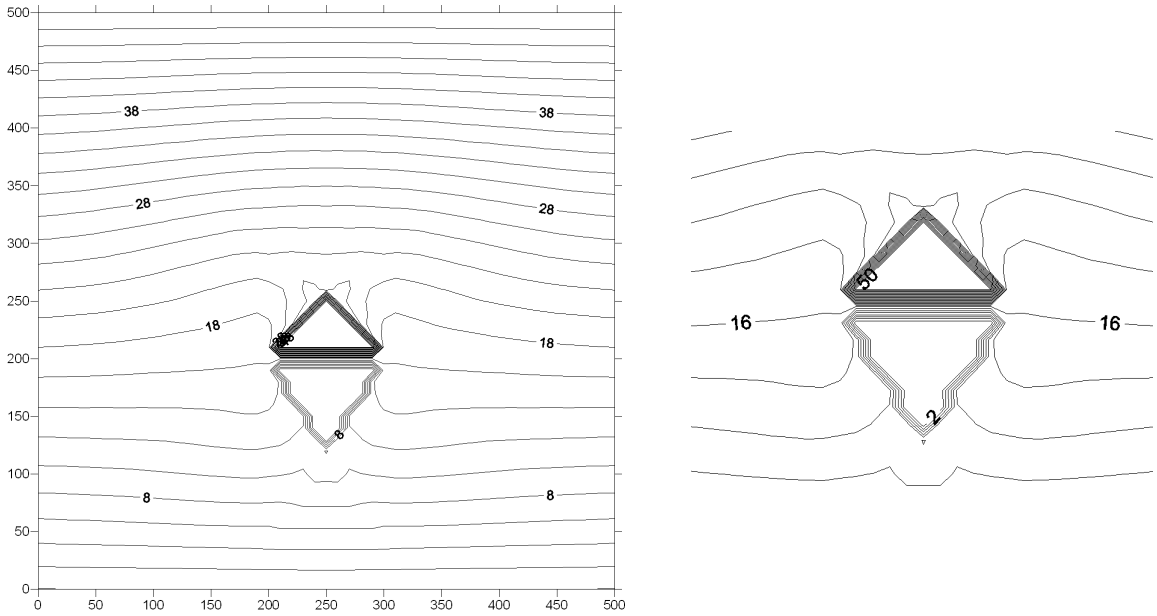


Figure 2.16: Representation of the screen effect.

It does reflect the screen effect as expected. However, the result is still quite different from the one computed with *Visual Modflow* (see figure 2.15). In fact, the cones on each side of the screen illustrate the lack of data for these points : the only head values selected in the kriging system here are the ones from the constant heads that are on the same side of the screen. And these constant head boundaries have one single value along the boundary. Thus the estimation in these cones can only take the value of these constant heads : $0\,m$ below the screen, and $50\,m$ above it.

To have a proper map of this study area, four head data points taken from the *Visual Modflow* modeling were therefore added on each side of the screen. The result is shown in figure 2.17. It still doesn't look exactly like the *Visual Modflow* map of figure 2.15. The result does look similar to the one computed for a low-transmissivity zone with the "Wall" package of *Visual Modflow*, that can create a thin low-transmissivity zone between two grid cells (see figure 2.18).

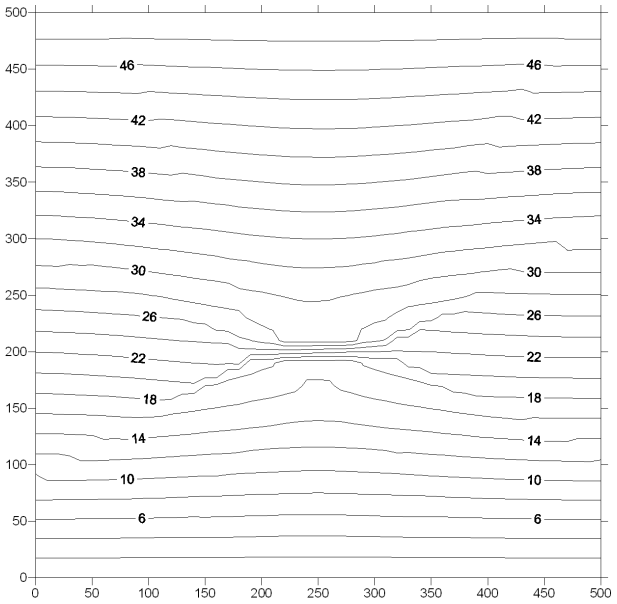


Figure 2.17: Screen effect with added data points.

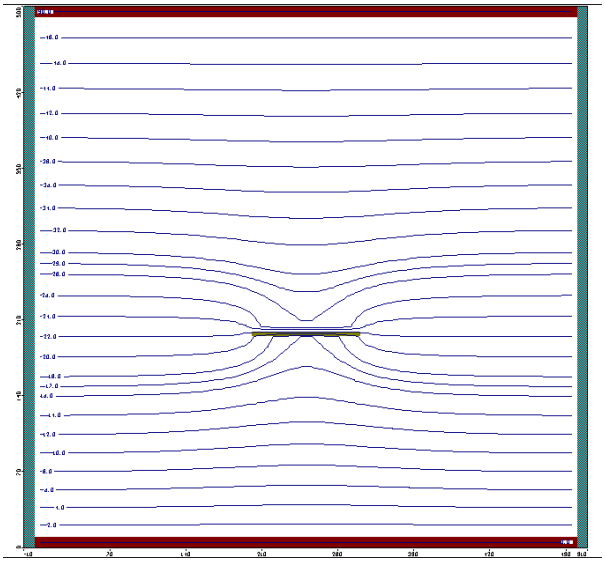


Figure 2.18: Visual Modflow modeled map. Wall package.

In order to prove that the added data points did not entirely solve the problem, the map presented in figure 2.19 shows the results with the added data points but without the screen effect.

And finally, the oblong no flow zone between $x = 200$ and $x = 300$, $y = 200$ and $y = 210$ modeled with *Visual Modflow* in figure 2.15 was introduced in Kriging under Boundary Conditions, with four no flow segments. The result is presented in figure 2.20. The result looks similar to figure 2.15. The only issue here is that the contouring algorithm preserves the continuity of the head, and thus the contour lines drawn cross the no flow zone. We would have to make them invisible in the no flow "box" to actually see the real result of Kriging under Boundary Conditions.

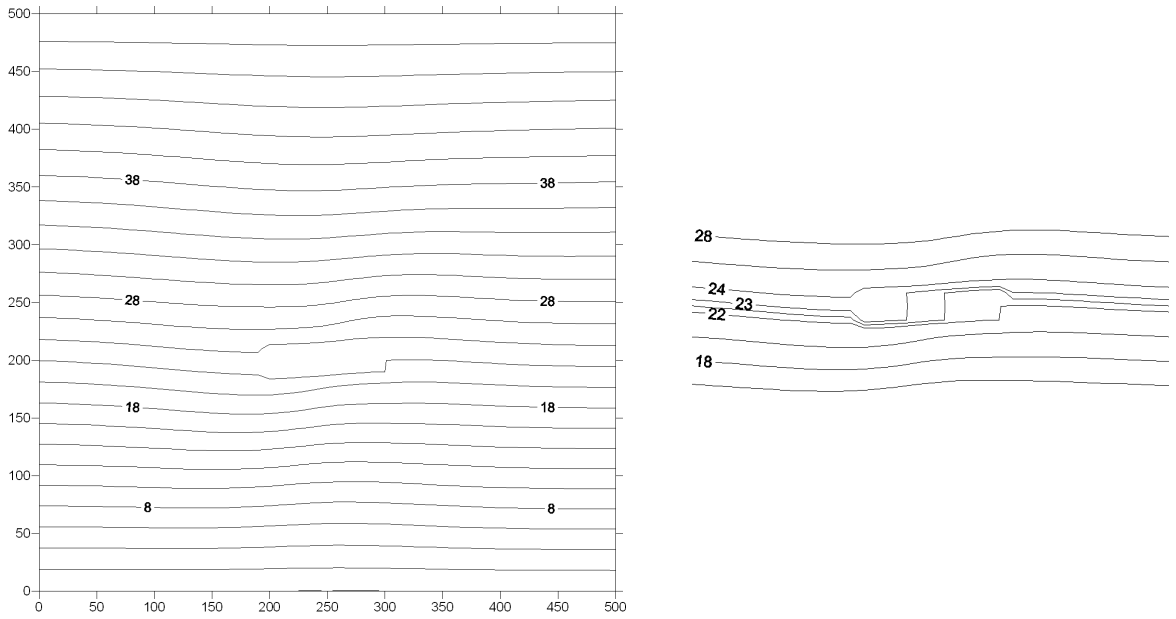


Figure 2.19: No flow boundary inside the study area, with added data points but without the screen effect.

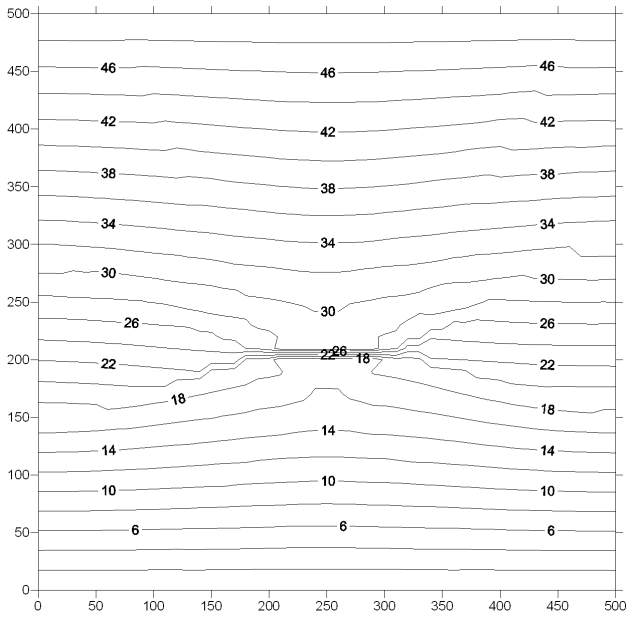


Figure 2.20: No flow "box" inside the study area.

2.5 Conclusion

Kriging under Boundary Conditions proved to be an efficient tool to take into account Dirichlet and Neumann conditions, without adding more requirements to compute the kriging system but adding the boundaries themselves. However, one has to be aware that it does not solve other types of boundary conditions than the Dirichlet and the Neumann ones. In particular, a well cannot be taken into account with this method.

Chapter 3

(Co-)Kriging with Multivariate Structural Analysis

Chapter 2 showed that kriging can take into account some usual boundary conditions used to solve the diffusivity equation (2.2). However, can we consider that all our estimates are verifying the same partial differential equation as the data ? Obviously not. The diffusivity equation depends on several variables (transmissivity T, total discharge Q and storage coefficient S), and our Kriging under Boundary Conditions only uses the head data to compute its estimates. This chapter explains how a multivariate structural analysis can help the estimate to try and better verify the same partial differential equation as the data, both for kriging and cokriging. The guiding idea and some of the results used are based on a Ph.D. thesis by *Dong* (1990). However, the chapter starts with some "generalized" definitions of what was explained in subsection 1.1.4. These definitions are needed to explain the theory in this chapter.

3.1 Further Geostatistical Definitions

3.1.1 Intrinsic Random Function of order k (IRF-k)

A random function $Z(x)$ is intrinsic of order k if for any allowable measure $\lambda \in \Lambda_k$ the random function :

$$Z_\lambda(x) = \sum_i \lambda_i \, Z(x_i + x) \tag{3.1}$$

is second-order stationary in $x \in \mathbb{R}^n$ and has a zero mean. This is equivalent to :

$$\begin{cases} E\big[Z_\lambda(x)\big] = 0 \\ E\big[Z_\lambda(x)\, Z_\lambda(y)\big] = K_\lambda(y-x) \qquad \forall x, y \in \mathbb{R}^n, \quad \lambda \in \Lambda_k \end{cases} \tag{3.2}$$

An IRF-k is simply a random function with stationary increments of order k. The usual intrinsic model described in subsection 1.1.4.3 corresponds to $k = 0$. Clearly, an IRF-k is also an IRF-$(k+1)$ and of any higher order, since $\Lambda_{k+1} \subset \Lambda_k$.

The condition that increments of order k have a zero mean is introduced for a simpler presentation and does not restrict generality. If these increments are stationary, their mean is

necessarily a polynomial of degree $k+1$ at most (c.f. *Matheron* (1973)), which is eliminated by regarding $Z(x)$ as an IRF-$(k+1)$.

As usual with random functions, it will be assumed that $Z(x)$ is continuous in the mean square sense, to extend the theory from the space Λ_k of discrete measures to the space M_k of measures with compact supports.

3.1.2 Generalized Covariance

Subsection 1.1.4 explained that the correlation structure of an SRF (or IRF-(-1)) is defined by its ordinary covariance function $C(h)$ and the correlation structure of an IRF (or IRF-0) is defined by its variogram $\gamma(h)$. In the same manner, when the stationarity assumptions are limited to generalized increments of order k (IRF-k)), what characterizes the correlation structure of $z(x)$ is a function called generalized covariance, denoted by $K(h)$.

For an IRF-k Z and any pair of measures $\lambda, \mu \in \Lambda_k$, the generalized covariance function $K(h)$ of Z, defined on $\mathbb{R}^n$ is defined by :

$$E\big[Z(\lambda)\, Z(\mu)\big] = \sum_i \sum_j \lambda_i\, \mu_j\, K(y_j - x_i) \tag{3.3}$$

If $\lambda = \mu$, we then have :

$$E\big[Z(\lambda)^2\big] = \sum_i \sum_j \lambda_i\, \lambda_j\, K(y_j - x_i) \qquad (\lambda \in \Lambda_k) \tag{3.4}$$

$K(h)$ is a symmetric function and is used just as an ordinary covariance function $C(h)$ and we also have the following property :

Theorem 1. *Any continuous IRF-k has a continuous generalized covariance $K(h)$. $K(h)$ is unique as an equivalence class, in the sense that any other generalized covariance is of the form $K(h) + Q(h)$, where $Q(h)$ is an even polynomial of degree $2k$ or less.*

A useful result is the relation between the ordinary covariance σ of Z and its generalized covariance K (see *Dong* (1990) and *Chilès and Delfiner* (1999)) :

$$\sigma(x, y) = K(y - x) + \sum_{l=1}^{p} a_l(y)\, f^l(x) + \sum_{l=1}^{p} a_l(x)\, f^l(y) \tag{3.5}$$

With :

- f^l a monomial of degree ≤ 1,
- p the number of monomials,
- a_l some continuous functions.

3.2 Kriging the Head using Transmissivity Knowledge

3.2.1 Hydrogeologic Context

In Chapter 2, we have considered the diffusivity equation (2.1) in the case Q was nil and T was constant over the study area. We will now consider that the transmissivity T and the head h depend on the 1D flow direction x, while the discharge is still assumed nil. We can then write :

$$\begin{aligned} div(T\, grad\, h) &= 0 \qquad (3.6)\\ \frac{\partial}{\partial x}\left(T\frac{\partial h}{\partial x}\right) &= 0 \\ \frac{\partial T}{\partial x}\frac{\partial h}{\partial x} + T\frac{\partial^2 h}{\partial x^2} &= 0 \\ \frac{\partial^2 h}{\partial x^2} &= -\frac{1}{T}\frac{\partial T}{\partial x}\frac{\partial h}{\partial x} \qquad \text{with } \frac{\partial h}{\partial x} = J = \text{hydraulic gradient} \\ \frac{\partial^2 h}{\partial x^2} &= J\frac{\partial (Log\, T)}{\partial x} \qquad (3.7) \end{aligned}$$

The strong restriction of the result (3.7) has to be remembered : this equation assumes that the flow is unidimensional and that the hydraulic gradient is a constant on the study area.

3.2.2 The Stochastic Equation $\Delta Z = Y$

Equation (3.7) can be more generally written $\Delta Z = Y$, and not only applied to hydrogeology. This equation is named the Poisson equation and represents the dependency between the studied variable Z — the hydraulic head h in our case — and a given source term Y — $J\frac{\partial (Log\, T)}{\partial x}$, i.e the constant hydraulic gradient multiplied by the derivative on x of $Log(T)$.

We can notice that we could have also considered solving equation (2.2) $\Delta h = \frac{Q}{T}$, which is also a Poisson equation. In that case, the Y source term would have been $\frac{Q}{T}$ and we could have solved the Poisson equation $\Delta h = \frac{Q}{T}$, with the assumption that T is constant in the study area. Thus, we would have had the variogram of the total discharge Q defining the variogram of h. However, in this thesis, the emphasis has been put on linking h with the transmissivity variations.

To consider solving this equation with a geostatistical method, we have first to assume that Y and Z are random functions of $\mathbb{R}^n$, with $n = 2$ in our case, and that $\Delta Z = Y$ is therefore considered as a "stochastic equation". The first question that then arises is that of the existence of a stochastic model compatible with this equation. *Matheron* (1971a) has stated the following theroem :

Theorem 2. *If Y is a continuous IRF-k of $\mathbb{R}^n$, there exists a unique twice differentiable IRF-$(k+2p)$ Z satisfying the differential equation $\Delta^p Z = Y$, which implies that :*
If Y is a continuous SRF (i.e. IRF-(-1)) of $\mathbb{R}^n$, there exists a unique twice differentiable

IRF-1 Z satisfying the differential equation $\Delta Z = Y$ and :
If Y is a continuous IRF(-2) of $\mathbb{R}^n$, there exists a unique twice differentiable IRF-0 Z satisfying the differential equation $\Delta Z = Y$.

3.2.3 Covariance Model

From this, we can derive the relationship between the generalized covariances of Z and of Y. To keep things simple, let us consider only the case where Y is a zero-mean SRF (it is still possible to come down to this case and add a polynomial of degree 1 with constant coefficients to the usual solution of Z). Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ be two points of $\mathbb{R}^n$. Denoting by $C(h)$ the stationary covariance of Y, we get :

$$C(y-x) = E\big[Y(x)\,Y(y)\big] = E\big[\Delta Z(x)\,\Delta Z(y)\big] = E\left[\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 Z(x)}{\partial x_i^2}\,\frac{\partial^2 Z(y)}{\partial y_j^2}\right]$$

With Z being twice differentiable, its nonstationary covariance $\sigma(x, y)$ is differentiable four times, and therefore :

$$C(y-x) = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2}{\partial x_i^2}\,\frac{\partial^2}{\partial y_j^2}\,E\big[Z(x)\,Z(y)\big] = \Delta_x\,\Delta_y\,\sigma(x,y) \tag{3.8}$$

where Δ_x is the Laplacian operator applied with respect to x. Combining equations (3.5) and (3.8), we get :

$$\Delta_x\,\Delta_y\,\sigma(x,y) = \Delta_x\,\Delta_y\,K(y-x) = \Delta^2\,K(h) \quad \text{with } h = y - x$$

The covariance C of Y and the generalized covariance K of Z, with Z and Y linked by the equation $\Delta Z = Y$ are thus related by the equation :

$$\Delta^2\,K(h) = C(h) \tag{3.9}$$

Theorem 2 then tells us that, Y being an SRF, Z will be an IRF-1.

However, the equation (3.7) we want to solve is more of the form $\Delta Z = \dfrac{\partial Y}{\partial x}$, with Z being the hydraulic head and Y being $Log(T)$. To study this equation knowing the result (3.9) of $\Delta Z = Y$, we will use a variable X defined by $Z = \dfrac{\partial X}{\partial x}$ (there is no real physical explanation for this variable).

We then have $\Delta\left(\dfrac{\partial X}{\partial x}\right) = \dfrac{\partial Y}{\partial x}$, which implies that $\dfrac{\partial}{\partial x}\,(\Delta X) = \dfrac{\partial Y}{\partial x}$. This is the derivative on x of the equation $\Delta X = Y$, in which Y is an SRF (or IRF-(-1)) with a covariance $C_Y(h)$ and X is an IRF-1 with a generalized covariance K so that we have, as in equation (3.9) :

$$\Delta^2\,K(h) = C_Y(h) \tag{3.10}$$

Z, the derivative of X on x, is then an IRF-0 and the relation between K and the variogram γ_H of the hydraulic head Z is :

$$\gamma_H(h) = J^2\,\frac{\partial^2 K}{\partial h_x^2}(h) \tag{3.11}$$

With h_x being the first coordinate of the vector $h(h_x, h_y)$. We can notice that the variogram of the variable Z is not isotropic and depends on the angle between the vector h and the flow direction.

3.2.4 Covariance Choice and Code Implementation

Dong (1990) computed the variograms γ_H of the hydraulic head h integrated from $\gamma_{Log(T)}$ of $Log(T)$ for several usual variogram models in $\mathbb{R}$, $\mathbb{R}^2$ and $\mathbb{R}^3$.

The calculations were made using several common variogram models for $Log(T)$. We have chosen to restrict ourselves to the spherical model as it is usually the one chosen for $Log(T)$. The corresponding model for H is therefore[1] :

$$\gamma_H(h) = \begin{cases} \frac{ca^2}{16}\left[\left(h_a^2 - \frac{8}{15}h_a^3 + \frac{8}{175}h_a^5\right) + dx_a^2\left(2 - \frac{8}{5}h_a + \frac{8}{35}h_a^3\right)\right] & \text{if } h_a \leq 1 \\ \frac{ca^2}{16}\left[\left(\frac{32}{75} + \frac{3}{35\,h_a^2} + \frac{4}{5}Log(h_a)\right) + dx_a^2\left(\frac{4}{5h_a^2} - \frac{6}{35h_a^4}\right)\right] & \text{if } h_a \geq 1 \end{cases} \tag{3.12}$$

With :

- c being the sill of the variogram of $Log(T)$,
- a being its range,
- $h_a = \dfrac{h}{a}$
- dx_a being the first coordinate of h_a in the Cartesian coordinate system in which the x axis is defined by the direction of the hydraulic gradient.

In the GSLIB algorithm, the covariance model associated to the variogram defined in (3.12) has been coded. To simplify the input for the user, instead of asking the sill of the variogram, the program asks to input the ratio $\dfrac{T_{max}}{T_{min}}$, and the sill c of the head variogram is then computed assuming :

$$\begin{aligned} Log\left(\frac{T_{max}}{T_{min}}\right) &= 4\sqrt{Var(Log\,T)} \\ Var(Log\,T) &= \frac{1}{16}\left[Log\left(\frac{T_{max}}{T_{min}}\right)\right]^2 \\ c &= J^2\,Var(Log\,T) \\ c &= \frac{J^2}{16}\left[Log\left(\frac{T_{max}}{T_{min}}\right)\right]^2 \end{aligned} \tag{3.13}$$

3.2.5 Application

The same example as the one used in subsection 2.4.1 has been computed, taking into account the full boundary conditions. The output map is presented in figure 3.1.

[1] If one manages to get his hand on Anne Dong's Ph.D thesis, p239, he might be surprised to find that there is no h_a^5 behind the term $\frac{8}{175}$. This is indeed a typo in Dong's thesis, as a constant term in a spherical based variogram is not correct and the h_a^5 term is needed to ensure the continuity of the first derivative of $\gamma(h)$.

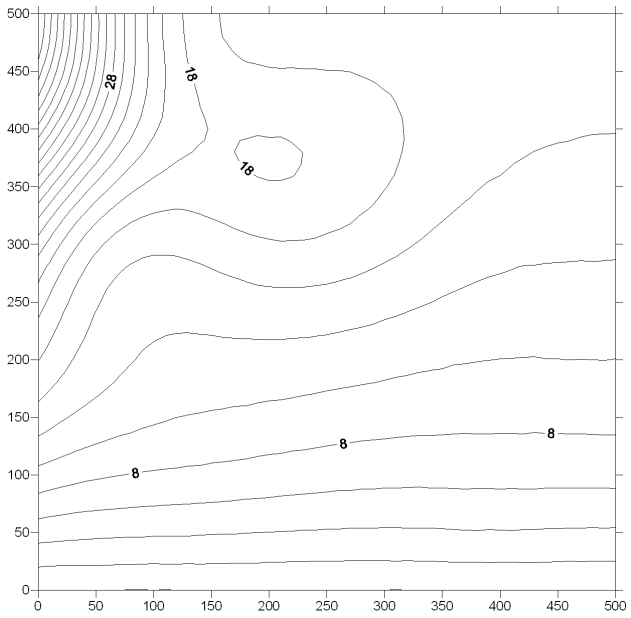


Figure 3.1: Kriging under Boundary Conditions with γ of h computed from γ of $Log(T)$.

There are no real improvements yet over the map presented in figure 2.7, except that the no flow boundary conditions are a bit better honored in the northeastern corner. However, using the $Log(T)$ integrated variogram was not supposed to drastically improve the Kriging under Boundary Conditions. It is more a step towards full cokriging between h and $Log(T)$.

3.3 Cokriging Head and Log Transmissivity

3.3.1 Cross-covariance

Before describing the cokriging system, let us complete the multivariate structural analysis of our study area. Thanks to section 3.2, we have linked the head variogram with the $Log(T)$ one. To compute the kriging system, we also have to know the cross-covariance between h and $Log(T)$ (see subsection 3.3.2).

First, we have to notice that $E\big[Z(x)\,Y(y)\big]$ doesn't necessarily exist. We have proved in subsection 3.2.3 that Z is an IRF-0 when Y is an SRF. The product $Z(x)\,Y(y)$ then has an hybrid status. We will make the hypothesis that we are in the "good" case and that $E\big[Z(x)\,Y(y)\big]$ exists. This is a common assumption for the hydraulic head.

As for the covariance in subsection 3.2.3, we have for the cross-covariance :

$$E\big[Z(x)\,Y(y)\big] = J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x,y) \tag{3.14}$$

For the spherical model in $\mathbb{R}^2$, we have[2] :

$$\frac{\partial \Delta K}{\partial \mathtt{x}}(x,y) = \begin{cases} c\,(x_1 - y_1)\,\left[\dfrac{1}{2} - \dfrac{1}{2}h_a + \dfrac{1}{10}h_a^3\right] & \text{if} \quad h_a \leq 1 \\ \dfrac{c\,(x_1 - y_1)}{10\,h_a^2} & \text{if} \quad h_a \geq 1 \end{cases} \tag{3.15}$$

[2]See *Dong* (1990), p243

With the same notations as for equation (3.12) and x_1, y_1 being the first coordinates of the points x and y in the Cartesian coordinate system in which the first axis `x` is defined by the direction of the hydraulic gradient.

3.3.2 The Cokriging System

As the kriging process has already been detailed twice in previous chapters, the demonstration is shortened as much as possible here. Basically, the process is almost identical to the one used for Kriging under Boundary Conditions, except that we have here a "true" secondary variable with cross-covariance terms and its own covariance terms, instead of the differences between head covariances we had in Kriging under Boundary Conditions. The system described is the Cokriging with a Trend (or Universal Cokriging) system, and it estimates the hydraulic head based on head and transmissivity data.

Z stands for the hydraulic head h and Y represents $Log(T)$ where T is the transmissivity. Z is defined as in previous chapters (see equations (1.22) and (1.23)), this time using the anistropic variogram $\gamma_H(h)$, and $Y(y) = m_Y + R_Y(y)$ where the mean m_Y is a constant and the covariance σ of Y is isotropic.

3.3.2.1 Linearity Constraint

$$Z^\star(x_0) = \lambda_0 + \sum_{i=1}^{N} \lambda_i \, Z(x_i) + \sum_{s=1}^{S} \theta_s \, Y(x_s)$$

3.3.2.2 Authorization Constraint

$$\begin{aligned} Z^\star(x_0) - Z(x_0) \;=\; & \underbrace{\lambda_0 + \sum_{i=1}^{N} \lambda_i \, m(x_i) - m(x_0)}_{\text{non random terms}} + \sum_{i=1}^{N} \lambda_i \, R(x_i) - R(x_0) \\ & + \sum_{s=1}^{S} \theta_s \, R_Y(x_s) \end{aligned}$$

Thus the authorization constraint is once again :

$$\lambda_0 + \sum_{i=1}^{N} \lambda_i \, m(x_i) - m(x_0) = 0 \tag{3.16}$$

3.3.2.3 Unbiasedness Constraint

$$\begin{aligned} E[Z^\star(x_0) - Z(x_0)] &= 0 \\ E\left[\lambda_0 + \sum_{i=1}^{N} \lambda_i\, Z_i + \sum_{s=1}^{S} \theta_s\, Y_s - Z_0\right] &= 0 \\ \lambda_0 + \sum_{i=1}^{N} \lambda_i\, m(x_i) - m(x_0) &= 0 \\ \lambda_0 + \sum_{l=0}^{L} a_l \left[\sum_{i=1}^{N} \lambda_i\, f_i^l - f_0^l\right] + \sum_{s=1}^{S} \theta_s\, m_Y &= 0 \end{aligned}$$

This is true if $\lambda_0 = 0$ and :

$$\forall l = 0, \ldots, L, \quad \sum_{i=1}^{N} \lambda_i\, f_i^l = f_0^l \qquad \text{and} \qquad \forall m_Y, \quad \sum_{s=1}^{S} \theta_s = 0 \tag{3.17}$$

The authorization constraint is *de facto* met. Once again, we have $f^0(x) = 1$, and the weights λ_i add up to 1 whereas the weights θ_s add up to zero.

3.3.2.4 Optimality Constraint

As for Universal Kriging, we have to minimize $Var[Z^\star(x_0) - Z(x_0)]$. However, in this case, the covariance of the hydraulic head represented here by Z is not strictly defined. We have to use its anisotropic variogram. The result presented in subsection 1.1.5 can be written in variogram terms :

$$E\Big[\big(Z(x_i) - Z(x_0)\big)\big(Z(x_j) - Z(x_0)\big)\Big] = -\gamma_{ij} + \gamma_{i0} + \gamma_{j0} \quad \text{with} \quad \gamma_{ij} = \gamma(x_i - x_j)$$

$$\begin{aligned} Var[Z^\star(x_0) - Z(x_0)] = & -\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i\lambda_j\, \gamma_{ij} + \sum_{s=1}^{S}\sum_{t=1}^{S} \theta_s\theta_t\, \sigma_{st} + 2\sum_{i=1}^{N}\sum_{s=1}^{S} \lambda_i\theta_s\, J\, \frac{\partial \Delta K}{\partial \mathtt{x}}(x_i, x_s) \\ & + 2\sum_{i=1}^{N} \lambda_i\, \gamma_{i0} - 2\sum_{s=1}^{S} \theta_s\, J\, \frac{\partial \Delta K}{\partial \mathtt{x}}(x_0, x_s) \end{aligned}$$

As for Kriging under Boundary Conditions, we obtain the derivatives respectively on λ_i, θ_s and μ_l to compute the kriging system of $N + S + L + 1$ linear equations with $N + S + L + 1$ unknowns :

$$\begin{cases} -\sum_{j=1}^{N} \lambda_j\, \gamma_{ij} + \sum_{s=1}^{S} \theta_s\, J\, \frac{\partial \Delta K}{\partial \mathtt{x}}(x_i, x_s) + \sum_{l=0}^{L} \mu_l\, f_i^l = -\gamma_{i0} & \forall i = 1, \ldots, N \\ \sum_{i=1}^{N} \lambda_i\, J\, \frac{\partial \Delta K}{\partial \mathtt{x}}(x_i, x_s) + \sum_{t=1}^{S} \theta_t\, \sigma_{st} + \mu_0 = J\, \frac{\partial \Delta K}{\partial \mathtt{x}}(x_0, x_s) & \forall s = 1, \ldots, S \\ \sum_{i=1}^{N} \lambda_i\, f_i^l = f_0^l & \forall l = 0, \ldots, L \\ \sum_{s=1}^{S} \theta_s = 0 & \end{cases} \tag{3.18}$$

In matrix notations, the kriging system (3.18) is of the following structure, with the usual notations :

$$\left(\begin{array}{c|c|c|c} -\gamma_{ij} & J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_i,x_t) & f_i^l & 0 \\ \hline J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_j,x_s) & \sigma_{st} & 0 & 1 \\ \hline f_j^l & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \end{array}\right) \left(\begin{array}{c} \lambda_j \\ \hline \theta_t \\ \hline \mu_l \\ \hline \mu_0 \end{array}\right) = \left(\begin{array}{c} -\gamma_{i0} \\ \hline J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_0,x_s) \\ \hline f_0^l \\ \hline 0 \end{array}\right)$$

The Universal Cokriging variance will then be :

$$\sigma^2_{UCoK} = \sum_{i=1}^{N} \lambda_i\,\gamma_{i0} - \sum_{s=1}^{S} \theta_s\,J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_0,x_s) - \sum_{l=0}^{L} \mu_l\,f_0^l \tag{3.19}$$

3.3.3 The Cokriging under Boundary Conditions System

The Cokriging under Boundary Conditions system can easily be obtained by combining the Kriging under Boundary Conditions system (2.9) and the Cokriging system (3.18). Only its matrix is represented there.

$$\left(\begin{array}{c|c|c|c|c} -\gamma_{ij} & -\gamma_{i\beta_1}+\gamma_{i\beta_2} & J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_i,x_t) & f_i^l & 0 \\ \hline \begin{array}{c}-\gamma_{j\alpha_1}\\+\\ \gamma_{j\alpha_2}\end{array} & \begin{array}{c}-\gamma_{\alpha_1\beta_1}+\gamma_{\alpha_1\beta_2}\\ \\ +\gamma_{\alpha_2\beta_1}-\gamma_{\alpha_2\beta_2}\end{array} & \begin{array}{c} J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_{\alpha 1},x_t)\\ -\\ J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_{\alpha 2},x_t)\end{array} & \begin{array}{c} f_{\alpha_1}^l\\ -\\ f_{\alpha_2}^l\end{array} & 0 \\ \hline J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_j,x_s) & \begin{array}{c} J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_{\beta 1},x_s)\\ -\\ J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_{\beta 2},x_s)\end{array} & \sigma_{st} & 0 & 1 \\ \hline f_j^l & f_{\beta_1}^l - f_{\beta_2}^l & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \end{array}\right) \left(\begin{array}{c} \lambda_j \\ \hline \xi_\beta \\ \hline \theta_t \\ \hline \mu_l \\ \hline \mu_0 \end{array}\right) = \left(\begin{array}{c} -\gamma_{i0} \\ \hline \begin{array}{c}-\gamma_{\alpha_1 0}\\+\\ \gamma_{\alpha_2 0}\end{array} \\ \hline J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_0,x_s) \\ \hline f_0^l \\ \hline 0 \end{array}\right)$$

The Cokriging under Boundary Conditions variance will then be :

$$\sigma^2_{UCoKBC} = \sum_{i=1}^{N} \lambda_i\,\gamma_{i0} + \sum_{\alpha=1}^{A} \xi_\alpha\left[\gamma_{\alpha_1 0} - \gamma_{\alpha_2 0}\right] - \sum_{s=1}^{S} \theta_s\,J\,\frac{\partial \Delta K}{\partial \mathtt{x}}(x_0,x_s) - \sum_{l=0}^{L} \mu_l\,f_0^l \tag{3.20}$$

3.4 Inverse Problem

Solving the partial differential equation $div(T\,grad\,h) = 0$ is called solving the direct problem, as the estimated variable is h, the main variable of the partial differential equation. Consequently, the inverse problem is estimating T, knowing h under zero discharge conditions ("virgin state" of the aquifer). In practice, it is a very common problem in hydrogeology that often has to be solved before modeling with different discharge conditions (e.g. wells).

To create a groundwater model, one has to enter the parameters of the partial differential equation, including the transmissivity. However, transmissivity data are scarce, while the hydraulic head is usually better known, because it only requires piezometer logging while estimating the transmissivity requires a heavier pumping test. Therefore solving the inverse problem gives us a better knowledge of T before modeling. This is critical as the better the transmissivity input, the easier it will be to obtain a good calibration when modeling, since it is easier to optimize parameters if they are closer from their real value since the beginning. A geostatistical approach makes it possible to take account of the joint spatial variability of h and T, thereby to restrict the space of possible equations and, in the end, to express the set of solutions as a family of conditional simulations. However, no one has really applied so far Anne Dong's results to use cokriging with a multivariate structural analysis based on $Log(T)$ for a practical problem.

3.4.1 The Inverse Problem Kriging System

The cokriging system for estimating the transmissivity is almost the same as the one used to compute the head (cf. equation (3.18)). In fact, the kriging matrix is identical to the one used for the direct problem. The variables, the data sets and the variographic parameters are the same. Only the estimated variable changes and so only the second term is indeed different. Thus the right-hand side matrices for Cokriging and Cokriging under Boundary Conditions will respectively be :

$$\left(\begin{array}{c} J\,\frac{\partial \Delta K}{\partial \mathbf{x}}(x_i, x_0) \\ \hline \sigma_{s0} \\ \hline 0 \\ \hline 1 \end{array}\right) \qquad \left(\begin{array}{c} J\,\frac{\partial \Delta K}{\partial \mathbf{x}}(x_i, x_0) \\ \hline J\,\frac{\partial \Delta K}{\partial \mathbf{x}}(x_{\alpha_1}, x_0) \\ - \\ J\,\frac{\partial \Delta K}{\partial \mathbf{x}}(x_{\alpha_2}, x_0) \\ \hline \sigma_{s0} \\ \hline 0 \\ \hline 1 \end{array}\right)$$

And the cokriging variances without and with boundary conditions will respectively be :

$$\sigma^2_{UCoK_{Inv}} = \sigma(0) - \sum_{i=1}^{N} \lambda_i \, J \, \frac{\partial \Delta K}{\partial \mathtt{x}}(x_i, x_0) - \sum_{s=1}^{S} \theta_s \, \sigma_{s0} - \mu_0 \tag{3.21}$$

$$\begin{aligned} \sigma^2_{UCoKBC_{Inv}} = \sigma(0) &- \sum_{i=1}^{N} \lambda_i J \frac{\partial \Delta K}{\partial \mathtt{x}}(x_i, x_0) - \sum_{\alpha=1}^{A} \xi_\alpha \left[J \, \frac{\partial \Delta K}{\partial \mathtt{x}}(x_{\alpha_1}, x_0) - J \, \frac{\partial \Delta K}{\partial \mathtt{x}}(x_{\alpha_2}, x_0) \right] \\ &- \sum_{s=1}^{S} \theta_s \, \sigma_{s0} - \mu_0 \end{aligned} \tag{3.22}$$

3.4.2 The Bias in Lognormal Kriging

In this chapter, we have studied two hydrogeologic variables considered as RFs : the hydraulic head h and the logarithm of the transmissivity $Log(T)$, but we never questioned the idea of applying kriging to the logarithm of a parameter. In fact, the variations of T are highly nonlinear. So it would not be wise to use the linear estimators of kriging on T itself. On the other hand, the logarithm of T can be considered as a Gaussian RF and thus be a good candidate for kriging. That had to be cleared.

However, when we solve the inverse problem, we want an estimate of T, not $Log(T)$. Is it possible to just compute the exponential of the $Log(T)$ estimate to obtain the T estimate ? In fact not, there is a correction factor to apply when computing the transmissivity estimate. Matheron already mentioned this when he first described kriging (*Matheron* (1963)), and revisited the concepts of so-called lognormal kriging (*Matheron* (1974)). For Ordinary Cokriging[3], the estimate will be for a RF $Z = Log(Y)$:

$$Z^\star(x_0) = exp\left[Y^\star(x_0) + \frac{\sigma^2_Y(x_0)}{2} - \mu_0 \right] \tag{3.23}$$

μ_0 being the Lagrange multiplier for Ypresent in the kriging system.

However, *Roth* (1998) believes that this estimator is still biased, and that we can't really get rid of the bias as long as we don't perform Simple Cokriging. In this thesis, we will present a map of $Log(T)$, thus avoiding this biasedness issue.

3.5 Application

This section details the application of solving either the direct, either the inverse problem, using the results and the system described in sections 3.3 and 3.4.

3.5.1 Code Implementation

GSLIB has a cokriging program, but it only does Simple or Ordinary Cokriging. The first task was to add the drift terms to compute the Universal Kriging system, using the

[3]The previously described system is called Universal Cokriging, but, in fact, we only take into account the drift for h, not for $Log(T)$, whose mean is unknown but assumed constant.

kriging algorithm. Then, the boundary conditions were added as in the matrix presented in subsection 3.3.3, and the cross-covariance model introduced in subsection 3.3.1 was coded. Finally, the right-hand term of the kriging system used to solve the inverse proble has been implemented, and a new input parameter allows to choose which variable the user wants to estimate.

3.5.2 Results

What differentiates our study from "classic" cokriging is the fact that we take account of the partial differential equation $\frac{\partial^2 h}{\partial x^2} = J \frac{\partial (Log\, T)}{\partial x}$ in our multivariate structural analysis. So the improvement will lie in the covariance and cross-covariance functions computed in sections 3.2.4 and 3.3.1. Let us have a closer look on these functions. Figure 3.2 shows their representation for an exponential covariance of $Log(T)$.

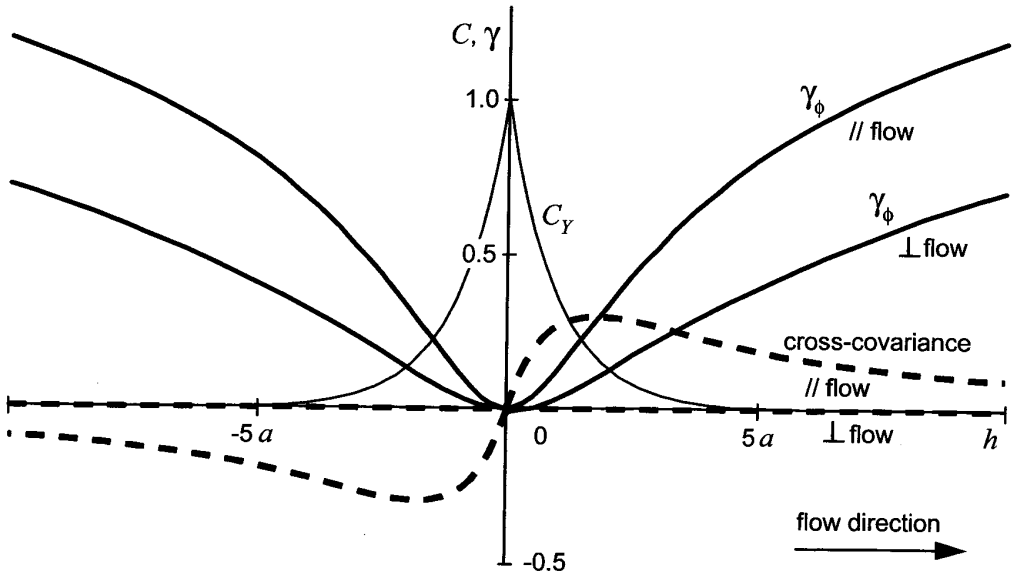


Figure 3.2: Exponential covariance C_Y of $Y = Log(T)$, variogram γ_ϕ of head perturbation ϕ and cross-covariance of $Y(x)$ and $\phi(x+h) - \phi(x)$ in the two-dimensional case, for an unidirectional flow in an infinite aquifer, from *Chilès and Delfiner* (1999), p.620.

In fact, the kriged estimate is a linear combination of weights that are functions of these covariances, translated to be centered at the estimated points. We have written that the product of the hydraulic gradient J and the range a of the $Log(T)$ variogram is involved linearly in the cross-covariance (3.14) and as a square in the head variogram (3.12). This means that in a cokriging approach for estimating the hydraulic head, the weights on $Log(T)$ data will be proportional to Ja, while when estimating the transmissivity, the weights on head data will be inversely proportional to Ja. Knowledge of this parameter is therefore essential, as it will determine by how much the head map will be distorted. a alone represents the range of the distortion.

Figures 3.3, 3.4 and 3.5 present the representations of the covariance of $Log(T)$ and both, the variogram of h and the cross-covariance between $Log(T)$ and h computed from this covariance. The flow is assumed parallel to the North-South axis.

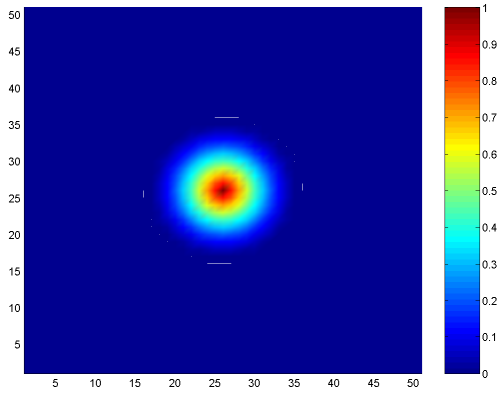


Figure 3.3: 2D representation of $C\big(Log(T)\big)$ centered on the point $(25; 25)$.

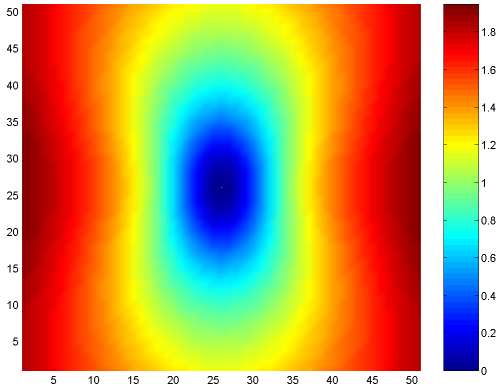


Figure 3.4: 2D representation of $\gamma(h)$ centered on the point $(25; 25)$.

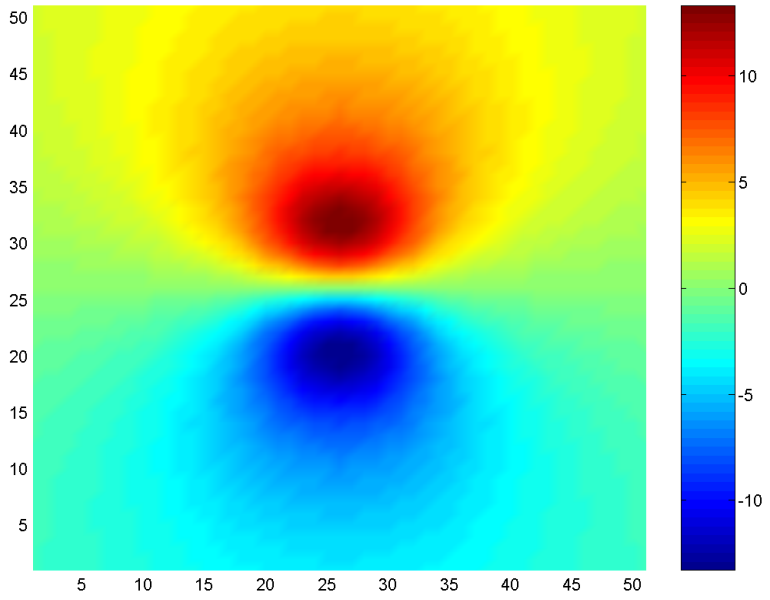


Figure 3.5: 2D representation of the cross-covariance between $Log(T)$ and h centered at point $(25; 25)$.

The antisymmetry of the cross-covariance (3.14) when computing the function around the central point has to be denoted. We can use it to make the following statements, assuming there is a low transmissivity zone :

- There is necessarily a point downstream that will be below this hydraulic head plane defined by the regional gradient J,
- And there is a point upstream that will be above this plane.

This leads to the conclusion that there is a higher hydraulic gradient at the location of the low transmissivity point.

The opposite statement can also be made if we assume there is a high transmissivity zone. We would then have a smaller hydraulic gradient at the location of the high transmissivity point.

Similarly, for the inverse problem, if we have, for example, a point whose head value is below the plane representing the regional hydraulic gradient, we can then conclude that the transmissivity values will be below the mean value upstram, and above it downstream. The opposite statement can be made for a head value below the regional plane. Figure 3.6 sums up these statements.

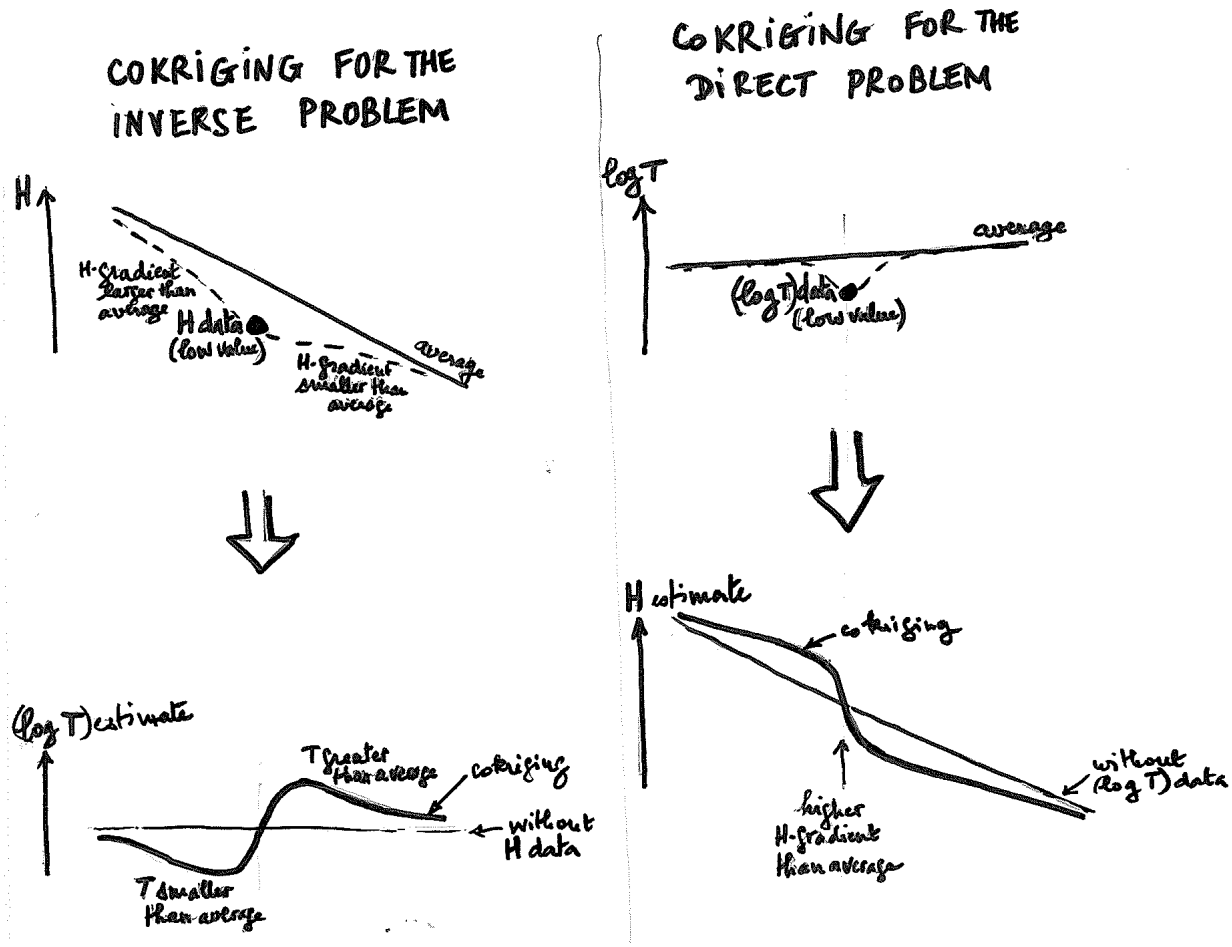


Figure 3.6: Summary of the influence of the cross-covariance anti-symmetry on h and $Log(T)$ estimates, by Jean-Pierre Delhomme.

A first example, with the same study area as usual, a constant head $h = 50\,m$ on the northern boundary and $h = 0\,m$ on the southern one and a low transmissivity point in the middle of the area is mapped in figure 3.7. A parameter that has to be noticed here is the range of the covariance of $Log(T)$. It is set to $70\,m$ in this example, as the boundary conditions must be "beyond the range" of $Log(T)$ data points.

The second example displayed in figure 3.8 presents an inverse problem. The study area and the parameters are the same as for the first example, except that instead of a low-transmissivity data point, we have two head data points in the middle of the area : one above the regional hydraulic gradient plane ($x = 250,\ y = 251,\ h = 26$), and the other below it ($x = 250,\ y = 249,\ h = 24$).

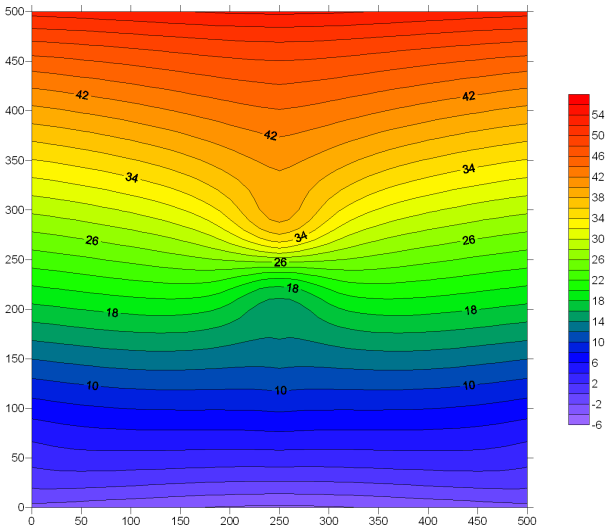


Figure 3.7: Cokriging with one low transmissivity point in the middle of the study area.

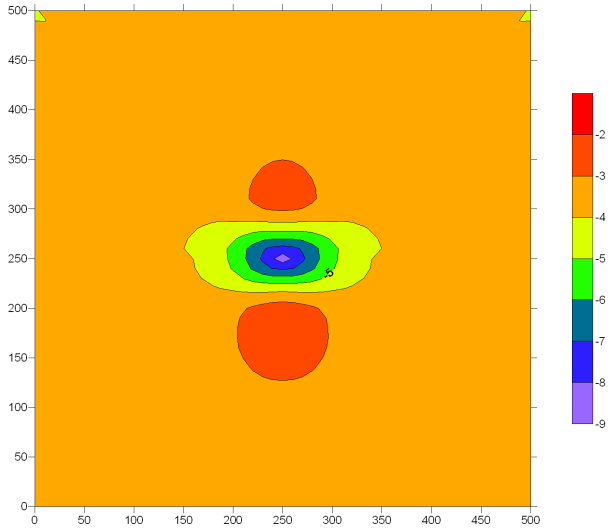


Figure 3.8: Inverse problem : $Log_{10}(T)$ map cokriged from h and $Log(T)$ data.

Finally, another interesting figure is figure 3.9. It presents the difference between the hydraulic head cokriged in figure 3.7 and the hydraulic head of the case in which there is no low-transmissivity data point and thus only the regional hydraulic gradient of $0.1\,m/m$ applies. This figure has some interesting similarities with figure 3.5[4]. This emphasizes the fact that it is the cross-covariance function which disrupts the hydraulic head map when we insert a low-tansmissivity point.

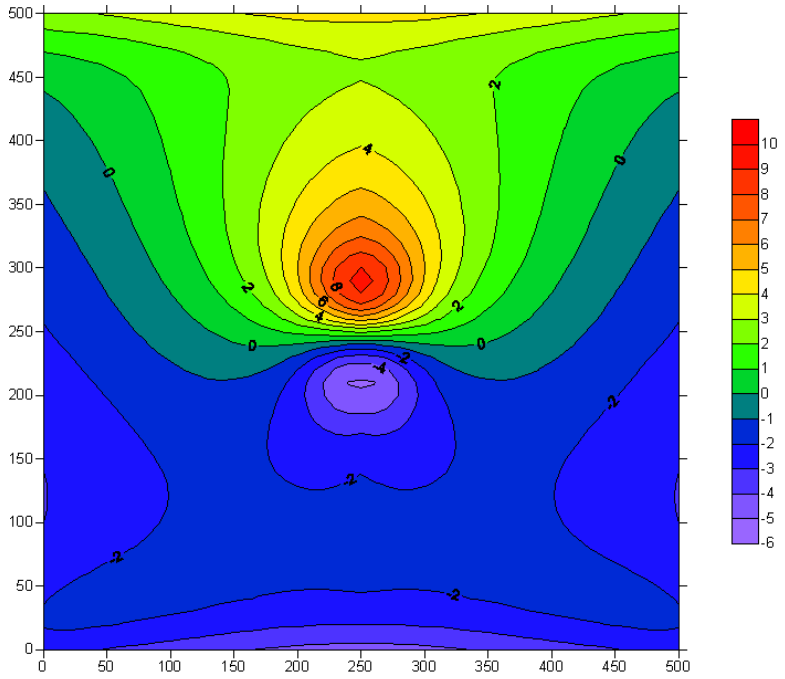


Figure 3.9: Difference between the hydraulic head maps with and without the low-transmissivity data point.

[4]Note that the hydraulic gradient is from East to West in figure 3.5 and from North to South in figure 3.9.

3.5.3 Conclusion

There is still some testing to be done in order to have a stable cokriging algorithm using this multivariate spatial analysis, but one can easily understand the potential that lies there, as shown in the very first example. This method could be used to solve both the direct and the inverse problem of the partial differential equation $\frac{\partial^2 h}{\partial x^2} = J \frac{\partial (Log\, T)}{\partial x}$.

Conclusion

The goal of this thesis was to make kriging respect some conditions of the partial differential equations that dictate the groundwater flow in hydrogeology. We have concluded in Chapter 2 that Kriging under Boundary Conditions allows us to take into account the Dirichlet and the Neumann boundary conditions. Kriging under Boundary Conditions proved to be robust enough to be easily used by a hydrogeologist. Consequently, it should be implemented soon in *GW Contour*. It seems a perfect tool to improve the results computed by this program that basically interpolates the hydraulic head, the hydraulic conductivity and the porosity and then uses these data to solve the Darcy equation on each node of the interpolation grid, in order to produce a velocity vector map and particle tracking.

Chapter 3 focused on making use of the partial differential equation $\frac{\partial^2 h}{\partial x^2} = J \frac{\partial (Log\, T)}{\partial x}$, in a multivariate spatial analysis cokriging approach, to take into account the transmissivity data and the fact that its structure is deeply linked to the hydraulic head structure. This is a more generalized problem than Kriging under Boundary Conditions, as the important hydrogeologic parameter that transmissivity is, is not considered constant anymore and is indeed used to evaluate the hydraulic head surface. This promising last minute research needs further work and testing, but the first results look very promising and the fact that it can also solve the inverse problem makes it even more interesting.

Bibliography

Surfer 8 User's Guide, Golden Software Inc., 2002.

GW Contour 1.0 - User's Manual, Waterloo Hydrogeologic Inc., 2005.

Intel Visual Fortran Compiler 9.0 Documentation, Intel Corporation, 2005a.

Visual MODFLOW 4.1 Professional Edition - User's Manual, Waterloo Hydrogeologic Inc., 2005b.

Arnaud, M., and X. Emery, *Estimation et interpolation spatiale*, Hermes Science Publications, Paris, 2000.

Baillargeon, S., Le krigeage : revue de la théorie et application à l'interpolation spatiale de données de précipitation, Master's thesis, Faculté des sciences et de génie, Université Laval, Québec, 2005.

Chauvet, P., *Processing data with a spatial support : Geostatistics and its methods*, Cahiers de Géostatistique, Fascicule 4, Ecole des Mines de Paris, 1993.

Chilès, J.-P., and P. Delfiner, *Geostatistics : Modeling Spatial Uncertainty*, Wiley series in probability and statistics, John Wiley & Sons, Inc., 1999.

Chilès, J.-P., and G. Matheron, Interpolation optimale et cartographie, *Annales des Mines*, pp. 1–7, 1975.

Cressie, N. A. C., *Statistics for spatial data*, Wiley series in probability and statistics, John Wiley & Sons, Inc., 1993.

Dagan, G., A note on the higher-order corrections of the head covariances in steady aquifer flow, *Water Resources Research*, *21*, 573–578, 1985.

Dagan, G., *Flow and Transport in Porous Formations*, Springer, Berlin Heidelberg, 1989.

de Marsily, G., *Hydrogéologie quantitative*, Masson, Paris, 1981.

de Marsily, G., *Quantitative Hydrogeology*, Academic Press Inc., 1986.

de Marsily, G., J.-P. Delhomme, A. Coudrain-Ribstein, and A. M. LaVenue, Four decades of inverse problems in hydrogeology, *Special Paper 348: Theory, modeling, and field investigation in hydrogeology: a special volume in honor of Shlomo P. Neumans 60th birthday*, *348*, 1–17, 2000.

Delhomme, J.-P., Applications de la théorie des variables régionalisées dans les sciences de l'eau, Ph.D. thesis, Université Pierre et Marie Curie - Paris VI, 1976.

Delhomme, J.-P., Kriging in the hydrosciences, *Advances in Water Resources*, *1*, 251–266, 1978.

Delhomme, J.-P., Kriging under boundary conditions, Presented at the American Geophysical Union Fall Meeting, San Francisco, 1979.

Deutsch, C. V., and A. G. Journel, *GSLIB : Geostatistical Software Library and User's Guide*, Applied Geostatistics Series, 2nd ed., Oxford University Press, 1998.

Dong, A., Estimation géostatistique des phénomènes régis par des équations aux dérivées partielles, Ph.D. thesis, Centre de Géostatistique, Ecole Nationale Supérieure des Mines de Paris, 1990.

Erickson, J., Computational geometry : Convex hull and plane sweep algorithms, *Lecture notes*, University of Illinois at Urbana-Champaign, 2002.

Gratton, Y., Le krigeage : la méthode optimale d'interpolation spatiale, *Les Articles de l'Institut d'Analyse Géographique*, pp. 1–4, 2002.

Hoekesma, R. J., and P. K. Kitanidis, An application of the geostatistical approach to the inverse problem in two-dimensional groundwater modeling, *Water Resources Research*, *20*, 1003–1020, 1984.

Jaquet, O., and S. Vomvoris, De l'apport de la géostatistique à la résolution du problème inverse : l'expérience de macro-perméabilité effectuée au laboratoire de Grimsel, Suisse, *Cahiers de Géostatistique, Fascicule 1, Ecole des Mines de Paris*, pp. 38–50, 1991.

Kitanidis, P. K., *Introduction to Geostatistics : Applications in Hydrogeology*, Cambridge University Press, 1997a.

Kitanidis, P. K., Comment on "A reassessment of the groundwater inverse problem" by D. McLaughlin and L. R. Townley, *Water Resources Research*, *33*, 2199–2202, 1997b.

Kitanidis, P. K., The minimum structure solution to the inverse problem, *Water Resources Research*, *33*, 2263–2272, 1997c.

Kitanidis, P. K., and E. G. Vomvoris, A geostatistical approach to the inverse problem in groundwater modeling (steady state) and one-dimensional simulations, *Water Resources Research*, *19*, 677–690, 1983.

LaVenue, A. M., B. S. RamaRao, G. de Marsily, and M. G. Marietta, Pilot point methodology for automated calibration of an ensemble of conditionally simulated transmissivity fields - 2. Application, *Water Resources Research*, *31*, 495–516, 1995.

Lent, T. V., and P. K. Kitanidis, Effects of first-order approximations on head and specific discharge covariances in high-contrast log conductivity, *Water Resources Research*, *32*, 1197–1207, 1996.

Mardyanto, M. A., and E. Evgin, A stochastic approach for the identification of hydraulic conductivity of a region, *Environmental Informatics Archives*, *1*, 99–113, 2003.

Matheron, G., *Traité de Géostatistique appliquée - Tome I*, Mémoires du Bureau de Recherches Géologiques et Minières, No. 14, Paris, 1962.

Matheron, G., *Traité de Géostatistique appliquée - Tome II : Le krigeage*, Mémoires du Bureau de Recherches Géologiques et Minières, No. 24, Paris, 1963.

Matheron, G., *Les variables régionalisées et leur estimation. Une application de la théorie des fonctions aléatoires aux Sciences de la Nature*, Masson, Paris, 1965.

Matheron, G., *Le krigeage universel*, Cahiers du Centre de Morphologie Mathématique de Fontainebleau, Fascicule 1, Ecole des Mines de Paris, 1969.

Matheron, G., *La théorie des variables régionalisées et ses applications*, Cahiers du Centre de Morphologie Mathématique de Fontainebleau, Fascicule 5, Ecole des Mines de Paris, 1970.

Matheron, G., La théorie des fonctions aléatoires intrinsèques généralisées, *Tech. Rep. N-252*, Centre de Géostatistique, Fontainebleau, France, 1971a.

Matheron, G., *The theory of regionalized variables and its applications*, Cahiers du Centre de Morphologie Mathématique de Fontainebleau, Fascicule 5, Ecole des Mines de Paris, 1971b.

Matheron, G., The intrinsic random functions and their applications, *Advances in Applied Probability*, *5*, 439–468, 1973.

Matheron, G., Effet proportionnel et lognormalité ou : Le retour du serpent de mer, *Tech. Rep. N-374*, Centre de Géostatistique, Fontainebleau, France, 1974.

Matheron, G., C. Roth, and C. de Fouquet, Modélisation et cokrigeage de la charge et de la transmissivité avec conditions aux limites à distance finie, *Cahiers de Géostatistique, Fascicule 3, Ecole des Mines de Paris*, pp. 61–76, 1993.

Page, C. G., *Professional Programmer's Guide to Fortran77*, Hyperion Books, 1988.

RamaRao, B. S., A. M. LaVenue, G. de Marsily, and M. G. Marietta, Pilot point methodology for automated calibration of an ensemble of conditionally simulated transmissivity fields - 1. Theory and computanional experiments, *Water Resources Research*, *31*, 475–493, 1995.

Rivoirard, J., Concepts et méthodes de la géostatistique, *Lecture notes*, Centre de Géostatistique, Ecole Nationale Supérieure des Mines de Paris, 1995.

Rivoirard, J., Cours de géostatistique multivariable, *Lecture notes*, Centre de Géostatistique, Ecole Nationale Supérieure des Mines de Paris, 2003.

Roth, C., Contribution de la géostatistique à la résolution du problème inverse en hydrogéologie, Ph.D. thesis, Centre de Géostatistique, Ecole Nationale Supérieure des Mines de Paris, 1995.

Roth, C., Is lognormal kriging suitable for local estimation?, *Mathematical Geology*, *30*, 999–1009, 1998.

Roth, C., J.-P. Chilès, and C. de Fouquet, Combining geostatistics and flow simulators to identify transmissivity, *Advances in Water Resources*, *21*, 555–565, 1998.

Rubin, Y., and G. Dagan, Stochastic identification of transmissivity and effective recharge in steady groundwater flow. 1. Theory, *Water Resources Research*, *23*, 1185–1192, 1987a.

Rubin, Y., and G. Dagan, Stochastic identification of transmissivity and effective recharge in steady groundwater flow. 2. Case study, *Water Resources Research*, *23*, 1193–1200, 1987b.

Wackernagel, H., Cours de géostatistique multivariable, *Lecture notes*, Centre de Géostatistique, Ecole Nationale Supérieure des Mines de Paris, 1993.

Appendix A

GSLIB Code : Main Algorithm

```
      program main
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C                                                                            %
C Copyright (C) 1996, The Board of Trustees of the Leland Stanford           %
C Junior University.  All rights reserved.                                   %
C                                                                            %
C The programs in GSLIB are distributed in the hope that they will be        %
C useful, but WITHOUT ANY WARRANTY.  No author or distributor accepts        %
C responsibility to anyone for the consequences of using them or for         %
C whether they serve any particular purpose or work at all, unless he        %
C says so in writing.  Everyone is granted permission to copy, modify        %
C and redistribute the programs in GSLIB, but only under the condition       %
C that this notice and the above copyright notice remain intact.             %
C                                                                            %
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c-----------------------------------------------------------------------------
c
c                CoKriging of a 3-D Rectangular Grid
c                ***********************************
c
c This program estimates the value of a "primary" variable with primary
c and secondary data.  The program could be modified to jointly predict
c primary and secondary data.
c
c
c
c-----------------------------------------------------------------------------
        USE DFLIB
        use dfwin
      include  'coktbc.inc'
      CHARACTER(1) key / 'A' /
c
c Read the Parameter File and the Data:
c
      call readparm
c
c Call coktbc to krige the grid:
c
      call coktbc
c
c Finished:
c
```

```
      write(*,9998) VERSION
 9998 format(/' COKTBC Version: ',f5.3, ' Finished'/)
      stop
      end



      subroutine readparm
c-----------------------------------------------------------------------
c
c                  Initialization and Read Parameters
c                  **********************************
c
c The input parameters and data are read, some quick error checking is
c performed, and the statistics of all the variables being considered
c are written to standard output.
c
c
c
c-----------------------------------------------------------------------
        USE DFLIB
      include  'coktbc.inc'
      parameter(MV=20)
      real     var(MV),av(MV),ss(MV)
      integer  ivrl(MV),nn(MV),whatest
      character datafl*500,outfl*500,dbgfl*500,secfl*500,str*500
      logical  testfl,linmod,posdef
      integer*4 surflong,npars,stat
      double precision surfdbl
        CHARACTER(1) key / 'A' /
c SURFER output file
c
c
c I/O units:
c
      lin  = 1
      lout = 2
      ldbg = 3
c
c Note VERSION number:
c
      write(*,9999) VERSION
 9999 format(/' COKTBC Version: ',f5.3/)
c
c Get the name of the parameter file - try the default name if no input:
c
c       write(*,*) 'Which parameter file do you want to use?'
c       read (*,'(a40)') str
      npars=iargc()
        if (npars.ge.1) then
          call getarg(1,str)
        else
        call fileopen(str)
        endif
      write(*,*) 'FILE OPENED ',str
      if(str(1:1).eq.' ')str='coktbc.par                                '
      inquire(file=str,exist=testfl)
      if(.not.testfl) then
```

```
            write(*,*) 'ERROR - the parameter file does not exist,'
            write(*,*) '       check for the file and try again  '
            write(*,*)
            if(str(1:20).eq.'coktbc.par          ') then
                  write(*,*) '       creating a blank parameter file'
                  call makepar
                  write(*,*)
            end if
            stop
      endif
      open(lin,file=str,status='OLD')
c
c Find Start of Parameters:
c
 1    read(lin,'(a4)',end=98) str(1:4)
      if(str(1:4).ne.'STAR') go to 1
c
c Read Input Parameters:
c
      read(lin,'(a)',err=98) datafl
c       call chknam(datafl,40)
      write(*,*) ' data file = ',datafl

      read(lin,*,err=98) nvr
      write(*,*) ' number of variables = ',nvr
      if(nvr.gt.MAXVAR) stop 'nvr is too big - modify .inc file'
      if(nvr.gt.2) stop 'can not use more than 1 secondary variable'

      read(lin,*,err=98) whatest
      write(*,*) ' estimated variable: 0=head, 1=transmissivity ',
     +           whatest
      if(whatest.lt.0.or.whatest.gt.1) stop ' Est. variable = 0 ou 1. '

      read(lin,*,err=98) ixl,iyl,izl,ikod,(ivrl(i),i=1,nvr)
      write(*,*) ' columns = ',ixl,iyl,izl,ikod,(ivrl(i),i=1,nvr)

      read(lin,*,err=98) tmin,tmax
      write(*,*) ' trimming limits = ',tmin,tmax

      read(lin,*,err=98) icolloc
      write(*,*) ' co-located cokriging flag = ',icolloc
      if(icolloc.eq.1) then
            write(*,*)
            write(*,*)' The co-located cokriging flag does not work.'
            write(*,*)' Modify the search and ndmaxs for co-located.'
            write(*,*)' The original intent was for the program to'
            write(*,*)' establish the variograms using a Markov model.'
            write(*,*)' You can do that outside the program.'
            write(*,*)
            write(*,*)' Note: the collocated cokriging file is not used'
            write(*,*)
            stop
      end if

      read(lin,'(a)',err=98) secfl
c       call chknam(secfl,40)
      write(*,*) ' collocated cokriging file = ',secfl
```

```
      read(lin,*,err=98) iclcol
      write(*,*) ' column for covariate = ',iclcol

      read(lin,*,err=98) idbg
      write(*,*) ' debug level = ',idbg

      read(lin,'(a)',err=98) dbgfl
c       call chknam(dbgfl,40)
      write(*,*) ' debug file = ',dbgfl
      write(*,*)
      write(*,*) ' Some input parameters are now echoed to debug file'
      write(*,*)
c       open(ldbg,file=dbgfl,status='UNKNOWN')

      read(lin,'(a)',err=98) outfl
c       call chknam(outfl,40)
      write(*,*) ' output file = ',outfl

      read(lin,*,err=98) nx,xmn,xsiz
      write(*,*) ' nx, xmn, xsiz = ',nx,xmn,xsiz

      read(lin,*,err=98) ny,ymn,ysiz
      write(*,*) ' ny, ymn, ysiz = ',ny,ymn,ysiz

      read(lin,*,err=98) nz,zmn,zsiz
      write(*,*) ' nz, zmn, zsiz = ',nz,zmn,zsiz

      read(lin,*,err=98) nxdis,nydis,nzdis
      write(*,*) ' nxdis,nydis,nzdis = ',nxdis,nydis,nzdis
      if((nxdis*nydis*nzdis).gt.MAXDIS) then
            write(*,*) 'ERROR COKTBC: Too many discretization points '
            write(*,*) '              Increase MAXDIS or lower n[xy]dis'
            stop
      endif

      read(lin,*,err=98) nborhood
      write(*,*) ' constant or moving neighborhood: ',nborhood
      if(nborhood.lt.0.or.nborhood.gt.1) stop ' Neighborhood = 0 ou 1. '

      read(lin,*,err=98) ndmin,ndmaxp,ndmaxg,ndmaxs
      write(*,*) ' ndmin,ndmaxp,ndmaxg,ndmaxs = ',ndmin,ndmaxp,ndmaxg,
     +           ndmaxs

      read(lin,*,err=98) radiusp,radius1,radius2
      write(*,*) ' primary search radii = ',radiusp,radius1,radius2
      if(radiusp.lt.EPSLON) stop 'radius must be greater than zero'
      radsqdp = radiusp * radiusp
      sanisp1 = radius1 / radiusp
      sanisp2 = radius2 / radiusp

      read(lin,*,err=98) radiuss,radius1,radius2
      write(*,*) ' secondary search radii = ',radiuss,radius1,radius2
      if(radiuss.lt.EPSLON) stop 'radius must be greater than zero'
      radsqds = radiuss * radiuss
      saniss1 = radius1 / radiuss
      saniss2 = radius2 / radiuss

      read(lin,*,err=98) sang1,sang2,sang3
```

```
      write(*,*) ' search anisotropy angles = ',sang1,sang2,sang3

      read(lin,*,err=98) ktype
      write(*,*) ' kriging type = ',ktype
      if(ktype.lt.0.or.ktype.gt.2) stop ' ERROR: invalid kriging type'

      read(lin,*,err=98) (idrif(i),i=1,9)
      write(*,*) ' drift terms = ',(idrif(i),i=1,9)

      read(lin,*,err=98) (vmean(i),i=1,nvr)
      write(*,*) ' variable means = ',(vmean(i),i=1,nvr)

      read(lin,*,err=98) Tmean
      write(*,*) ' Tmean: ',Tmean

      read(lin,*,err=98) gradh,angh
      write(*,*) ' gradh,angh: ',gradh,angh

c Read Output File option
      read(lin,*) noutfile
        str=''
        read(lin,'(A500)',END=4) str
c Read whether user wants to interpolate log of values for each variable
      read(lin,*) logopt1,logopt2
      if(logopt1.lt.0.or.logopt1.gt.1) stop ' Log option = 0 ou 1. '
      if(logopt2.lt.0.or.logopt2.gt.1) stop ' Log option = 0 ou 1. '
      write(*,*)
c Read whether user wants to bound results within a max and a min
      read(lin,*) nrestmin,restmin
      read(lin,*) nrestmax,restmax
      if(nrestmin.ne.0.and.nrestmax.ne.0.and.restmin.gt.restmax) stop
     +       ' Restmin < Restmax ! '
c
c Now, initialize nst value to -1 to flag all missing variograms:
c
      do i=1,nvr
            do j=1,nvr
                  ind = i + (j-1)*MAXVAR
                  nst(ind) = -1
            end do
      end do
c
c Read as many variograms as are in the parameter file:
c
 3    read(lin,*,end=4,err=98) i,j
      if(i.gt.MAXVAR.or.j.gt.MAXVAR) then
            write(*,*) ' Variogram specified for variable beyond MAXVAR'
            stop
      end if
      ind = i + (j-1)*MAXVAR
      read(lin,*,err=98) nst(ind),c0(ind)
      write(ldbg,103) i,j,nst(ind),c0(ind)
      istart = 1 + (ind-1)*MAXNST
      do i=1,nst(ind)
            index = istart + i - 1
            read(lin,*,err=98) it(index),cc(index),ang1(index),
     +                         ang2(index),ang3(index)
            read(lin,*,err=98) aa(index),aa1,aa2
```

```
            anis1(index) = aa1 / max(aa(index),EPSLON)
            anis2(index) = aa2 / max(aa(index),EPSLON)
            if(it(index).eq.4.and.ktype.eq.0)
     +                      stop 'No Power model with SK'
          end do
          write(ldbg,104) (it(istart+i-1),   i=1,nst(ind))
          write(ldbg,105) (aa(istart+i-1),   i=1,nst(ind))
          write(ldbg,106) (cc(istart+i-1),   i=1,nst(ind))
          write(ldbg,107) (ang1(istart+i-1), i=1,nst(ind))
          write(ldbg,108) (ang2(istart+i-1), i=1,nst(ind))
          write(ldbg,109) (ang3(istart+i-1), i=1,nst(ind))
          write(ldbg,110) (anis1(istart+i-1),i=1,nst(ind))
          write(ldbg,111) (anis2(istart+i-1),i=1,nst(ind))
 103      format(/,' USER input variogram for variables ',i2,' and ',i2,/,
     +           '       number of structures=',i2,' nugget effect=',f12.4)
 104      format('       types of structures: ',10i2)
 105      format('       aa values:           ',10f12.4)
 106      format('       cc values:           ',10f12.4)
 107      format('       ang1 values:         ',10f12.4)
 108      format('       ang2 values:         ',10f12.4)
 109      format('       ang3 values:         ',10f12.4)
 110      format('       anis1 values:        ',10f12.4)
 111      format('       anis2 values:        ',10f12.4)
          go to 3
 4        close(lin)
          write(*,*)
c
c Fill in cross variograms j=i if they have not been explicitly entered:
c
          do i=1,nvr
          do j=1,nvr
                ind1 = i + (j-1)*MAXVAR
                ind2 = j + (i-1)*MAXVAR
                if(nst(ind1).eq.-1.and.nst(ind2).eq.-1) then
                      write(*,*) ' Need variogram between variables ',i,j
                      stop
                end if
                if(nst(ind1).eq.-1) then
                      nst(ind1) = nst(ind2)
                      c0(ind1)  = c0(ind2)
                      istart1   = 1 + (ind1-1)*MAXNST
                      istart2   = 1 + (ind2-1)*MAXNST
                      do ist=1,nst(ind1)
                            index2        = istart2 + ist - 1
                            index1        = istart1 + ist - 1
                            it(index1)    = it(index2)
                            cc(index1)    = cc(index2)
                            aa(index1)    = aa(index2)
                            ang1(index1)  = ang1(index2)
                            ang2(index1)  = ang2(index2)
                            ang3(index1)  = ang3(index2)
                            anis1(index1) = anis1(index2)
                            anis2(index1) = anis2(index2)
                      end do
                else if(nst(ind2).eq.-1) then
                      nst(ind2) = nst(ind1)
                      c0(ind2)  = c0(ind1)
                      istart1   = 1 + (ind1-1)*MAXNST
```

```
                  istart2    = 1 + (ind2-1)*MAXNST
                  do ist=1,nst(ind2)
                        index2         = istart2 + ist - 1
                        index1         = istart1 + ist - 1
                        it(index2)     = it(index1)
                        cc(index2)     = cc(index1)
                        aa(index2)     = aa(index1)
                        ang1(index2)   = ang1(index1)
                        ang2(index2)   = ang2(index1)
                        ang3(index2)   = ang3(index1)
                        anis1(index2)  = anis1(index1)
                        anis2(index2)  = anis2(index1)
                  end do
            end if
      end do
      end do
c
c Rescale the "Tmax/Tmin" parameter to make it equal to the sill
c
      if(nvr.eq.2.or.it(1).eq.7) then
         do i=1,MXVARG*MAXNST
            if(cc(i).gt.0) then
               cc(i) = (0.25 * LOG(cc(i)))**2
            else if(cc(i).lt.0) then
               write(*,*) ' Warning: nil or negative sill defined '
            endif
         end do
      end if
c
c Has the linear model of coregionalization been used?
c
      linmod = .true.
      do i=1,nvr
      do j=1,nvr
            ind1 = i + (j-1)*MAXVAR
            do i2=1,nvr
            do j2=1,nvr
                  ind2 = i2 + (j2-1)*MAXVAR
                  if(nst(ind1).ne.nst(ind2)) linmod = .false.
                  istart1 = 1 + (ind1-1)*MAXNST
                  istart2 = 1 + (ind2-1)*MAXNST
                  do ist=1,nst(ind1)
                     index2 = istart2 + ist - 1
                     index1 = istart1 + ist - 1
                     if(it(index1).ne.it(index2).or.
     +                abs(aa(index1)    - aa(index2)).gt.EPSLON.or.
     +                abs(ang1(index1)  - ang1(index2)).gt.EPSLON.or.
     +                abs(ang2(index1)  - ang2(index2)).gt.EPSLON.or.
     +                abs(ang3(index1)  - ang3(index2)).gt.EPSLON.or.
     +                abs(anis1(index1) - anis1(index2)).gt.EPSLON.or.
     +                abs(anis2(index1) - anis2(index2)).gt.EPSLON)
     +                linmod = .false.
                  end do
            end do
            end do
      end do
      end do
      if(linmod) then
```

```
c
c Yes, the linear model of coregionalization has been used, now check
c to ensure positive definiteness:
c
            posdef = .true.
            do i=1,nvr
            do j=1,nvr
               if(i.ne.j) then
                  ii = i+(i-1)*MAXVAR
                  jj = j+(j-1)*MAXVAR
                  ij = i+(j-1)*MAXVAR
                  ji = j+(i-1)*MAXVAR
                  istartii = 1 + (ii-1)*MAXNST
                  istartjj = 1 + (jj-1)*MAXNST
                  istartij = 1 + (ij-1)*MAXNST
                  istartji = 1 + (ji-1)*MAXNST
c
c First check the nugget effects:
c
                  if(c0(ii).le.0.0.or.c0(jj).le.0.0.or.
     +              (c0(ii)*c0(jj)).lt.(c0(ij)*c0(ji)) ) then
                        posdef = .false.
                        write(ldbg,120) i,j
                  endif
                  do ist=1,nst(ii)
                        indexii = istartii + ist - 1
                        indexjj = istartjj + ist - 1
                        indexij = istartij + ist - 1
                        indexji = istartji + ist - 1
                        if(cc(indexii).le.0.0.or.cc(indexjj).le.0.0.or.
     +                    (cc(indexii)*cc(indexjj)).lt.
     +                    (cc(indexij)*cc(indexji)) ) then
                              posdef = .false.
                              write(ldbg,121) ist,i,j
                        endif
                  end do
               end if
            end do
            end do
 120        format(/,'Positive definiteness violation on nugget effects'
     +           ,/,' between ',i2,' and ',i2)
 121        format(/,'Positive definiteness violation on structure ',i2
     +           ,/,' between ',i2,' and ',i2)
c
c The model is not positive definite:
c
c            if(.not.posdef) then
c            write(*,*)
c            write(*,*) ' The linear model of coregionalization is NOT'
c            write(*,*) ' positive definite! This could lead to singular'
c            write(*,*) ' matrices and unestimated points.'
c            write(*,*)
c            write(*,*) ' Do you want to proceed? (y/n)'
c            read (*,'(a)') str
c            if(str(1:1).ne.'y'.and.str(1:1).ne.'Y') stop
c            end if
c      else
c
```

```
c No linear model of coregionalization:
c
c             write(*,*)
c             write(*,*) ' A linear model of coregionalization has NOT'
c             write(*,*) ' been used!!  This could lead to many singular '
c             write(*,*) ' matrices and unestimated points.'
c             write(*,*)
c             write(*,*) ' Do you want to proceed? (y/n)'
c             read (*,'(a)') str
c             if(str(1:1).ne.'y'.and.str(1:1).ne.'Y') stop
      endif
c Open Surfer files
      if (str.ne.'') then
                if (noutfile.eq.1) then
                        open(21,file=str,status='unknown')
                        write(21,'(a4)')'DSAA'
                        write(21,*) nx,ny
                        write(21,*) xmn,xmn+(nx-1)*xsiz
                        write(21,*) ymn,ymn+(ny-1)*ysiz
                endif
                if (noutfile.eq.2) then
                        open(21,file=str,form='binary',status='new')
                        write(21)'DSBB'
                        write(21) INT2(nx),INT2(ny)
                        write(21) dble(xmn),dble(xmn+(nx-1)*xsiz),dble(ymn),
     +                  dble(ymn+(ny-1)*ysiz)
                endif
        else
                if (noutfile.eq.1) then
                        call fopensurf(str)
                        open(21,file=str,status='unknown')
                        write(21,'(a4)')'DSAA'
                        write(21,*) nx,ny
                        write(21,*) xmn,xmn+(nx-1)*xsiz
                        write(21,*) ymn,ymn+(ny-1)*ysiz
                endif
                if (noutfile.eq.2) then
                        call fopensurf(str)
                        open(21,file=str,form='binary',status='new')
                        write(21)'DSBB'
                        write(21) INT2(nx),INT2(ny)
                        write(21) dble(xmn),dble(xmn+(nx-1)*xsiz),dble(ymn),
     +                  dble(ymn+(ny-1)*ysiz)
                endif
                if (noutfile.eq.3) then
                        call fopensurf(str)
          endif
          open(21,file=str,form='binary',status='unknown')
          surflong = 1112691524
          write(21) surflong
          surflong = 4
          write(21) surflong
          surflong = 1
          write(21) surflong
          surflong = 1145655879
          write(21) surflong
          surflong = 72
          write(21) surflong
```

```
            surflong = ny
            write(21) surflong
            surflong = nx
            write(21) surflong
            write(21) dble(xmn),dble(ymn),dble(xsiz),dble(ysiz)
      endif
c
c Perform some quick error checking:
c
      if(ndmin .le.0)       stop ' NDMIN too small'
      if(ndmaxp.gt.MAXSAM) stop ' NDMAXP too large'
      if(ndmaxg.gt.MAXSAM) stop ' NDMAXG too large'
      if(ndmaxs.gt.MAXSAM) stop ' NDMAXS too large'
      if((ndmaxs/2).le.nvr.and.ktype.eq.2) then
            write(*,100) nvr,ndmaxs
 100        format('WARNING: with traditional ordinary cokriging the ',
     +           /,'sum of the weights applied to EACH secondary data',
     +           /,'is zero.  With ndmaxs set low and nvr large the',
     +           /,'secondary data will not contribute to the estimate')
      endif
c
c Check to make sure the data file exists, then either read in the
c data or write an error message and stop:
c
      inquire(file=datafl,exist=testfl)
      if(.not.testfl) then
            write(*,*) 'ERROR data file ',datafl,' does not exist!'
            stop
      endif
c
c The data file exists so open the file and read in the header
c information. Initialize the storage that will be used to summarize
c the data found in the file:
c
      open(lin,file=datafl,status='OLD')
      read(lin,'(a)',err=99) str
      read(lin,*,err=99)       nvari
      do i=1,nvari
            read(lin,'()',err=99)
      end do
      do i=1,nvr
            nn(i) = 0
            av(i) = 0.0
            ss(i) = 0.0
      end do
c
c Some tests on column numbers:
c
      if(ixl.gt.nvari.or.iyl.gt.nvari.or.izl.gt.nvari.or.
     +   ikod.gt.nvari.or.ivrl(1).gt.nvari) then
            write(*,*) 'There are only ',nvari,' columns in input data'
            write(*,*) '  your specification is out of range'
            stop
      end if
c
c Read all the data until the end of the file:
c
      nd = 0
```

```
 7     read(lin,*,end=9,err=99) (var(j),j=1,nvari)
       nd = nd + 1
       if(nd.gt.MAXDAT) then
             write(*,*) ' ERROR: Exceeded available memory for data'
             stop
       end if
c
c Store data values (all secondary data must be transformed such that
c their mean is the same as the primary variable (if the first type of
c ordinary kriging is being used)):
c
       vr(nd) = var(ivrl(1))
       if(vr(nd).ge.tmin.and.vr(nd).lt.tmax) then
             nn(1) = nn(1) + 1
             av(1) = av(1) + vr(nd)
             ss(1) = ss(1) + vr(nd)*vr(nd)
       endif
       if (logopt1.eq.1) then
          if (vr(nd).gt.0.0) then
            vr(nd) = log(vr(nd))
            else
              vr(nd) = -9999999
              write(*,*)' Logarithmic Interpolation cannot be used for ',
     +                  'values <=0: is this value a no-data flag?'
            endif
       end if
       if(nvr.ge.2) then
             sec1(nd) = var(ivrl(2))
             if(sec1(nd).ge.tmin.and.sec1(nd).lt.tmax) then
                   nn(2) = nn(2) + 1
                   av(2) = av(2) + sec1(nd)
                   ss(2) = ss(2) + sec1(nd)*sec1(nd)
             endif
             if (logopt2.eq.1) then
                if (sec1(nd).gt.0.0) then
                  sec1(nd) = log(sec1(nd))
                  else
                    sec1(nd) = -9999999
                    write(*,*)' Logarithmic Interpolation cannot be used for ',
     +                        'values <=0: is this value a no-data flag?'
                  endif
             end if
       end if
c
c Assign the coordinate location of this data:
c
       if(ixl.le.0) then
             x(nd) = xmn
       else
             x(nd) = var(ixl)
       endif
       if(iyl.le.0) then
             y(nd) = ymn
       else
             y(nd) = var(iyl)
       endif
       if(izl.le.0) then
             z(nd) = zmn
```

```
      else
            z(nd) = var(izl)
      endif
      if(ikod.le.0) then
            kod(nd) = 0
      else
            kod(nd) = var(ikod)
      endif
      go to 7
 9    close(lin)
c
c Compute the averages and variances as an error check for the user:
c
      do i=1,nvr
            av(i) = av(i) / max(real(nn(i)),1.0)
            ss(i) =(ss(i) / max(real(nn(i)),1.0)) - av(i) * av(i)
            write(*,*) 'COKTBC Variable ',i,' in data file: ',ivrl(i)
            write(*,*) '  Number   = ',nn(i)
            write(*,*) '  Average  = ',av(i)
            write(*,*) '  Variance = ',ss(i)
      end do
c
c Create arrays for no flow screen segments
c
      call scrarr(nd,x,y,kod,vr,nsc)
      call screens(nd,x,y,kod,vr,nsc,xs1,ys1,xs2,ys2)
      write(113,*) 'Nb screen segments', nsc
      write(113,*) '  i       xs1        ys1        xs2       ys2'
      do i=1,nsc
        write(113,'(i4,4f10.2)') i,xs1(i),ys1(i),xs2(i),ys2(i)
      enddo
c
c Add points along boundary lines
c
      write(114,*) 'Nb points', nd
      csiz = (xsiz + ysiz) / 2.
      call bdarr(nd,x,y,kod,csiz,newnd)
      nd = newnd
      write(114,*) 'New nb points', nd
      write(114,*) '  i        x          y          z          ',
     +             'vr        sec1       kod'
        call bdpts(nd,x,y,z,vr,sec1,kod,csiz)
        do i=1,nd
        write(114,'(i4,5f10.2,i8)') i,x(i),y(i),z(i),vr(i),sec1(i),
     +                              kod(i)
      enddo
c
c Apply the correction factor for the constant flux data : Q -> Delta h
c
      call fluxcorr(nd,x,y,vr,kod,csiz,Tmean)
c
c Add 2 fictive x & y coordinates along prescribed flux boundary lines
c
        call ficcoord(nd,x,y,kod,csiz,ddx,ddy)
        write(115,*) '  i        x          y          vr         ',
     +               'sec1       kod      ddx        ddy'
        do i=1,nd
        write(115,'(i4,4f10.2,i8,2f10.2)') i,x(i),y(i),vr(i),sec1(i),
```

```
     +                                          kod(i),ddx(i),ddy(i)
      enddo
c
c Check For duplicate points
c
         call remdup(nd,x,y,z,vr,sec1,kod,ddx,ddy,newnd)
         write(*,*) nd,newnd
         write(*,*)'Duplicate X,Y Pairs removed =',nd-newnd
         if (newnd.lt.nd) nd = newnd
         write(1151,*) '  i        x          y          vr        ',
     +                 'sec1       kod       ddx        ddy'
         do i=1,nd
         write(1151,'(i4,4f10.2,i8,2f10.2)') i,x(i),y(i),vr(i),sec1(i),
     +                                       kod(i),ddx(i),ddy(i)
      enddo
c
c Open output files and write headers:
c
      open(lout,file=outfl,status='UNKNOWN')
      write(lout,101) str
 101  format('COKTBC with:',a40,/,'2',/,'estimate',/,
     +       'estimation variance')
      write(ldbg,102) str
 102  format(/,'DEBUGGING COKTBC with:',a40)
      return
c
c Error in an Input File Somewhere:
c
 98   stop 'ERROR in parameter file!'
 99   stop 'ERROR in data file!'
      end



      subroutine coktbc
c-----------------------------------------------------------------------
c
c                  CoKriging of a 3-D Rectangular Grid
c                  ***********************************
c
c This subroutine estimates point or block values of one variable by
c ordinary cokriging using up to MAXVAR variables.
c
c
c
c Original:  A.J. Desbarats                                         1984
c Head/Log(T) + BC Cokriging Add-On: J.P. Delhomme & P. Le Cointe    2006
c-----------------------------------------------------------------------
      include  'coktbc.inc'
      parameter(PMX=999.)
      real       distp(MAXSAM),dists(MAXSAM)
      real*8      cbb
      real(4), allocatable :: krigout(:)
      integer    nump(MAXSAM),nums(MAXSAM),vars(MAXSAM),whatest
      real*8      rottemp,blnkval
      integer*4   dataid,datalen
      logical    fircon
      data       fircon/.true./
```

```
      integer(4) :: nalloc
        nalloc = nx*ny
      allocate (krigout(nalloc))
        datamax = 1.0e-39
        datamin = 1.0e29
c
c Set up the search and covariance rotation matrices:
c
      covmax = c0(1)
      do is=1,nst(1)
            call setrot(ang1(is),ang2(is),ang3(is),anis1(is),anis2(is),
     +                  is,MAXROT,rotmat)
            if(it(is).eq.4) then
                  covmax = covmax + PMX
            else if(it(is).eq.7) then
                  ct = cc(is) * (gradh * aa(is) / 4.)**2
                  covmax = covmax + 20.*ct
            else if(it(is).eq.8) then
                  covmax = covmax + cc(is)*gradh
            else
                  covmax = covmax + cc(is)
            endif
      end do
      isrot = MAXNST + 1
      if(whatest.eq.0) then
            call setrot(sang1,sang2,sang3,sanisp1,sanisp2,isrot,MAXROT,
     +                  rotmat)
      else
            call setrot(sang1,sang2,sang3,saniss1,saniss2,isrot,MAXROT,
     +                  rotmat)
      endif
c
c Finish computing the rescaling factor and stop if unacceptable:
c
      if(radsqdp.lt.1.0) then
            resc = 2.0 * radiusp / max(covmax,0.0001)
      else
            resc =(4.0 * radsqdp)/ max(covmax,0.0001)
      endif
      if(resc.le.0.0) then
            write(*,*) 'ERROR KT3D: The rescaling value is wrong ',resc
            write(*,*) '            Maximum covariance: ',covmax
            write(*,*) '            search radius:      ',radiusp
            stop
      endif
      resc = 1.0 / resc
c
c Set up for super block searching:
c
      nsec = nvr - 1
      write(*,*) 'Setting up super block search strategy'
      call setsupr(nx,xmn,xsiz,ny,ymn,ysiz,nz,zmn,zsiz,nd,x,y,z,
     +             vr,ddx,ddy,tmp,nsec,sec1,MAXSBX,MAXSBY,MAXSBZ,nisb,
     +             nxsup,xmnsup,xsizsup,nysup,ymnsup,ysizsup,nzsup,
     +             zmnsup,zsizsup)
      write(116,*) '  i        x          y          z         ',
     +             'vr        sec1       ddx        ddy'
          do i=1,(nd)
```

```
          write(116,'(i4,7f10.2)') i,x(i),y(i),z(i),vr(i),sec1(i),
     +                             ddx(i),ddy(i)
        enddo
        call picksup(nxsup,xsizsup,nysup,ysizsup,nzsup,zsizsup,
     +               isrot,MAXROT,rotmat,radsqdp,nsbtosr,ixsbtosr,
     +               iysbtosr,izsbtosr)
c
c Compute the number of drift terms, if SK is being considered
c then we will set all the drift terms off and mdt to 0):
c
      mdt = 1
      do i=1,9
            if(ktype.eq.0.or.ktype.eq.1) idrif(i) = 0
            if(idrif(i).lt.0.or.idrif(i).gt.1) then
                  write(*,*) 'ERROR KT3D: invalid drift term',idrif(i)
                  stop
            endif
            mdt = mdt + idrif(i)
      end do
      if(ktype.eq.0) mdt = 0
      if(ktype.eq.1) mdt = 0
c
c Set up the discretization points per block.  Figure out how many
c are needed, the spacing, and fill the xdb, ydb and zdb arrays with
c the offsets relative to the block center (this only gets done once):
c
      ndb  = nxdis * nydis * nzdis
      xdis = xsiz  / max(real(nxdis),1.0)
      ydis = ysiz  / max(real(nydis),1.0)
      zdis = zsiz  / max(real(nzdis),1.0)
      xloc = -0.5*(xsiz+xdis)
      i    = 0
      do ix =1,nxdis
            xloc = xloc + xdis
            yloc = -0.5*(ysiz+ydis)
            do iy=1,nydis
                  yloc = yloc + ydis
                  zloc = -0.5*(zsiz+zdis)
                  do iz=1,nzdis
                        zloc = zloc + zdis
                        i = i+1
                        xdb(i) = xloc + 0.5*xsiz
                        ydb(i) = yloc + 0.5*ysiz
                        zdb(i) = zloc + 0.5*zsiz
                  end do
            end do
      end do
c
c Initialize accumulators:
c
      uk   = 0.0
      vk   = 0.0
      nk   = 0
c
c Calculate Block Covariance for head and eventually for transmissivity.
c Check for point kriging.
c
      call cova3(xdb(1),ydb(1),zdb(1),xdb(1),ydb(1),zdb(1),1,nst,MAXNST,
```

```
     +            c0,it,cc,aa,gradh,angh,1,MAXROT,rotmat,cmax,cova)
      unbias = dble(cova)
      if(whatest.ne.0) call cova3(xdb(1),ydb(1),zdb(1),xdb(1),ydb(1),
     +                            zdb(1),4,nst,MAXNST,c0,it,cc,aa,gradh,
     +                            angh,1,MAXROT,rotmat,cmax,cova)
      if(ndb.le.1) then
            cbb = cova
      else
            cbb = 0.0
            do i=1,ndb
              do j=1,ndb
                   if(whatest.eq.0) then
                        call cova3(xdb(i),ydb(i),zdb(i),xdb(j),ydb(j),
     +                             zdb(j),1,nst,MAXNST,c0,it,cc,aa,gradh,
     +                             angh,1,MAXROT,rotmat,cmax,cova)
                   else
                        call cova3(xdb(i),ydb(i),zdb(i),xdb(j),ydb(j),
     +                             zdb(j),4,nst,MAXNST,c0,it,cc,aa,gradh,
     +                             angh,1,MAXROT,rotmat,cmax,cova)
                   endif
                   if(i.eq.j) cova = cova - c0(1)
                   cbb = cbb + cova
              end do
            end do
            cbb = cbb/real(ndb*ndb)
      endif
      write(ldbg,*) 'Block average covariance ',cbb
c
c Mean values of the drift functions:
c
      do i=1,9
            bv(i) = 0.0
      end do
      xloc = -0.5*(xsiz+xdis)
      i    = 0
      do i=1,ndb
            bv(1) = bv(1) + xdb(i)
            bv(2) = bv(2) + ydb(i)
            bv(3) = bv(3) + zdb(i)
            bv(4) = bv(4) + xdb(i)*xdb(i)
            bv(5) = bv(5) + ydb(i)*ydb(i)
            bv(6) = bv(6) + zdb(i)*zdb(i)
            bv(7) = bv(7) + xdb(i)*ydb(i)
            bv(8) = bv(8) + xdb(i)*zdb(i)
            bv(9) = bv(9) + ydb(i)*zdb(i)
      end do
      do i=1,9
            bv(i) = (bv(i) / real(ndb)) * resc
      end do
c
c MAIN LOOP OVER ALL THE BLOCKS IN THE GRID:
c
      ncells = 0
      do 4 iz=1,nz
      zloc = zmn + (iz-1)*zsiz
      do 4 iy=1,ny
      yloc = ymn + (iy-1)*ysiz
      do 4 ix=1,nx
```

```
      xloc = xmn + (ix-1)*xsiz
c
c Find the nearest head data samples:
c
      call srchsupr(xloc,yloc,zloc,radsqdp,isrot,MAXROT,rotmat,nsbtosr,
     +              ixsbtosr,iysbtosr,izsbtosr,noct,nd,x,y,z,ddx,ddy,
     +              tmp,nisb,nxsup,xmnsup,xsizsup,nysup,ymnsup,ysizsup,
     +              nzsup,zmnsup,zsizsup,nsc,xs1,ys1,xs2,ys2,nclose,
     +              close,infoct)

c
c Load the nearest head data in xa,ya,za,vra,ddxa,ddya:
c
      np = 0
      na = 0
      do i=1,nclose
            if(np.eq.ndmaxp.and.nborhood.ne.0) go to 32
            ind = int(close(i)+0.5)
            if((vr(ind).ge.tmin).and.(vr(ind).lt.tmax).
     +         and.(np.lt.ndmaxp.or.nborhood.eq.0)) then
                  np = np + 1
                  na = na + 1
                  xa(na)   = x(ind) - xloc + 0.5*xsiz
                  ya(na)   = y(ind) - yloc + 0.5*ysiz
                  za(na)   = z(ind) - zloc + 0.5*zsiz
                  vra(na)  = vr(ind)
                  ddxa(na) = 0.0
                  ddya(na) = 0.0
                  iva(na)  = 1
            end if
      end do
c
c Test number of data samples found:
c
      if(np.lt.ndmin) then
            est  = UNEST
            estv = UNEST
            go to 4
      end if
c
c Test if there are enough data samples to estimate all drift terms:
c
      if(np.ge.1.and.np.le.mdt) then
            if(fircon) then
                  write(ldbg,999)
                  fircon = .false.
            end if
            est  = UNEST
            estv = UNEST
            go to 4
      end if
 999  format(' Encountered a location where there were too few data ',/,
     +       ' to estimate all of the drift terms but there would be',/,
     +       ' enough data for OK or SK.   KT3D currently leaves ',/,
     +       ' these locations unestimated.',/,
     +       ' This message is only written once - the first time.',/)
c
c Find the nearest "gradient" samples:
```

```
c
      call srchsupr2(xloc,yloc,zloc,radsqdp,isrot,MAXROT,rotmat,
     +               nsbtosr,ixsbtosr,iysbtosr,izsbtosr,noct,nd,x,y,z,
     +               ddx,ddy,tmp,nisb,nxsup,xmnsup,xsizsup,nysup,
     +               ymnsup,ysizsup,nzsup,zmnsup,zsizsup,nsc,xs1,ys1,
     +               xs2,ys2,nclose2,close2,infoct2)
c
c Load the nearest "gradient" data in xa,ya,za,vra,ddxa,ddya:
c
      ng = 0
      do i=1,nclose2
            if(ng.eq.ndmaxg.and.nborhood.ne.0) go to 32
            ind = int(close2(i)+0.5)
            if((vr(ind).ge.tmin).and.(vr(ind).lt.tmax).
     +         and.(ng.lt.ndmaxg.or.nborhood.eq.0)) then
                ng = ng + 1
                na = na + 1
                xa(na)   = x(ind) - xloc + 0.5*xsiz
                ya(na)   = y(ind) - yloc + 0.5*ysiz
                za(na)   = z(ind) - zloc + 0.5*zsiz
                vra(na)  = vr(ind)
                ddxa(na) = ddx(ind)
                ddya(na) = ddy(ind)
                iva(na)  = 1
                if(vr(ind).ne.0) then
                      ng = ng + 1
                      na = na + 1
                      xa(na)   = x(ind) - xloc + 0.5*xsiz
                      ya(na)   = y(ind) - yloc + 0.5*ysiz
                      za(na)   = z(ind) - zloc + 0.5*zsiz
                      vra(na)  = 0
                      ddxa(na) = -ddy(ind)
                      ddya(na) = ddx(ind)
                      iva(na)  = 1
                end if
            end if
      end do
c
c Find the nearest samples:
c
      call srchsupr3(xloc,yloc,zloc,radsqdp,isrot,MAXROT,rotmat,nsbtosr,
     +               ixsbtosr,iysbtosr,izsbtosr,noct,nd,x,y,z,tmp,nisb,
     +               nxsup,xmnsup,xsizsup,nysup,ymnsup,ysizsup,nzsup,
     +               zmnsup,zsizsup,nclose3,close3,infoct3)
c
c Load secondary data until maximum is met:
c
      ns = 0
      do i=1,nclose3
            if(ns.eq.ndmaxs.and.nborhood.ne.0) go to 32
            ind = int(close3(i)+0.5)
            if((sec1(ind).ge.tmin).and.(sec1(ind).lt.tmax).
     +         and.(nvr.ge.2).and.(ns.lt.ndmaxs.or.nborhood.eq.0)) then
                ns = ns + 1
                na = na + 1
                xa(na)   = x(ind) - xloc + 0.5*xsiz
                ya(na)   = y(ind) - yloc + 0.5*ysiz
                za(na)   = z(ind) - zloc + 0.5*zsiz
```

```
                  vra(na)  = sec1(ind)
                  ivar     = 2
                  if(ktype.ne.2)
     +            vra(na)  = vra(na) - vmean(ivar) + vmean(1)
                  ddxa(na) = 0.0
                  ddya(na) = 0.0
                  iva(na)  = 2
            end if
      end do
 32   continue
      write(119,*) '  i        x          y          z         ',
     +             'vr        ddx        ddy'
        do i=1,na
      write(119,'(i4,6f10.2)') i,xa(i),ya(i),za(i),vra(i),
     +                         ddxa(i),ddya(i)
      enddo
c
c Solve the Kriging System:
c
      if(ktype.eq.0) neq = na
      if(ktype.eq.1) neq = na + 1
      if(ktype.eq.2) neq = na + mdt + nvr - 1
      if((neq-na).gt.na.or.na.lt.ndmin) then
            write(lout,100) UNEST,UNEST
            go to 4
      end if
c
c Set up kriging matrices:
c
      do i=1,neq*neq
            a(i) = 0.0
      end do
      do i=1,neq
            r(i) = 0.0
      end do
      do j=1,na
            do i=1,j
                  ind = iva(i) + (iva(j)-1)*MAXVAR
                  if((i.gt.np).and.(i.le.np+ng).and.(j.gt.np).
     +               and.(j.le.np+ng)) then
                     xi1 = xa(i) - ddxa(i)
                     xi2 = xa(i) + ddxa(i)
                     yi1 = ya(i) - ddya(i)
                     yi2 = ya(i) + ddya(i)
                     xj1 = xa(j) - ddxa(j)
                     xj2 = xa(j) + ddxa(j)
                     yj1 = ya(j) - ddya(j)
                     yj2 = ya(j) + ddya(j)
                     call cova3(xi1,yi1,za(i),xj1,yj1,za(j),ind,
     +                          nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                          MAXROT,rotmat,cmax,cov1)
                     call cova3(xi2,yi2,za(i),xj1,yj1,za(j),ind,
     +                          nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                          MAXROT,rotmat,cmax,cov2)
                     call cova3(xi1,yi1,za(i),xj2,yj2,za(j),ind,
     +                          nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                          MAXROT,rotmat,cmax,cov3)
                     call cova3(xi2,yi2,za(i),xj2,yj2,za(j),ind,
```

```
     +                                  nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                                  MAXROT,rotmat,cmax,cov4)
                        cova = (cov1-cov2) - (cov3-cov4)
                  else if((i.le.np.and.j.gt.np.and.j.le.np+ng).or.
     +                    (i.gt.np.and.i.le.np+ng.and.j.gt.np+ng)) then
                        xj1 = xa(j) - ddxa(j)
                        xj2 = xa(j) + ddxa(j)
                        yj1 = ya(j) - ddya(j)
                        yj2 = ya(j) + ddya(j)
                        call cova3(xa(i),ya(i),za(i),xj1,yj1,za(j),
     +                             ind,nst,MAXNST,c0,it,cc,aa,gradh,
     +                             angh,1,MAXROT,rotmat,cmax,cov1)
                        call cova3(xa(i),ya(i),za(i),xj2,yj2,za(j),
     +                             ind,nst,MAXNST,c0,it,cc,aa,gradh,
     +                             angh,1,MAXROT,rotmat,cmax,cov2)
                        cova = cov1 - cov2
                  else
                        call cova3(xa(i),ya(i),za(i),xa(j),ya(j),
     +                             za(j),ind,nst,MAXNST,c0,it,cc,aa,
     +                             gradh,angh,1,MAXROT,rotmat,cmax,cova)
                  end if
                  a(neq*(i-1)+j) = dble(cova)
                  a(neq*(j-1)+i) = dble(cova)
            end do
            xx = xa(j)
            yy = ya(j)
            zz = za(j)
c
c Right hand side covariance if est. variable = head:
c
         if(whatest.eq.0) then
            iv  = 1
            ind = iv + (iva(j)-1)*MAXVAR
            if(ndb.le.1) then
                  if((j.le.np).or.(j.gt.np+ng)) then
                        call cova3(xx,yy,zz,xdb(1),ydb(1),zdb(1),ind,
     +                             nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                             MAXROT,rotmat,cmax,cova)
                        cb = cova
                  else
                        xi1 = xx - ddxa(j)
                        xi2 = xx + ddxa(j)
                        yi1 = yy - ddya(j)
                        yi2 = yy + ddya(j)
                        call cova3(xi1,yi1,zz,xdb(1),ydb(1),zdb(1),ind,
     +                             nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                             MAXROT,rotmat,cmax,cb1)
                        call cova3(xi2,yi2,zz,xdb(1),ydb(1),zdb(1),ind,
     +                             nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                             MAXROT,rotmat,cmax,cb2)
                        cb = cb1 - cb2
                  end if
            else
                  cb  = 0.0
                  do j1=1,ndb
                        if((j.le.np).or.(j.gt.np+ng)) then
                              call cova3(xx,yy,zz,xdb(j1),ydb(j1),
     +                                   zdb(j1),ind,nst,MAXNST,c0,it,cc,aa,
```

```
     +                                  gradh,angh,1,MAXROT,rotmat,cmax,cova)
                        else
                              xi1 = xx - ddxa(j)
                              xi2 = xx + ddxa(j)
                              yi1 = yy - ddya(j)
                              yi2 = yy + ddya(j)
                              call cova3(xi1,yi1,zz,xdb(1),ydb(1),
     +                              zdb(1),ind,nst,MAXNST,c0,it,cc,aa,
     +                              gradh,angh,1,MAXROT,rotmat,cmax,cov1)
                              call cova3(xi2,yi2,zz,xdb(1),ydb(1),
     +                              zdb(1),ind,nst,MAXNST,c0,it,cc,aa,
     +                              gradh,angh,1,MAXROT,rotmat,cmax,cov2)
                              cova = cov1 - cov2
                        end if
                        dx = xx - xdb(j1)
                        dy = yy - ydb(j1)
                        dz = zz - zdb(j1)
                        if((dx*dx+dy*dy+dz*dz).lt.EPSLON) then
                              cb = cb + cova - c0(ind)
                        else
                              cb = cb + cova
                        end if
                  end do
                  cb = cb / real(ndb)
            endif
            r(j)  = dble(cb)
c
c Right hand side covariance if est. variable = transmittivity:
c
         else
            iv  = 2
            ind = iv + (iva(j)-1)*MAXVAR
            if(ndb.le.1) then
                  if((j.le.np).or.(j.gt.np+ng)) then
                        call cova3(xdb(1),ydb(1),zdb(1),xx,yy,zz,ind,
     +                           nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                           MAXROT,rotmat,cmax,cova)
                        cb = cova
                  else
                        xi1 = xx - ddxa(j)
                        xi2 = xx + ddxa(j)
                        yi1 = yy - ddya(j)
                        yi2 = yy + ddya(j)
                        call cova3(xdb(1),ydb(1),zdb(1),xi1,yi1,zz,ind,
     +                           nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                           MAXROT,rotmat,cmax,cb1)
                        call cova3(xdb(1),ydb(1),zdb(1),xi2,yi2,zz,ind,
     +                           nst,MAXNST,c0,it,cc,aa,gradh,angh,1,
     +                           MAXROT,rotmat,cmax,cb2)
                        cb = cb1 - cb2
                  end if
            else
                  cb  = 0.0
                  do j1=1,ndb
                        if((j.le.np).or.(j.gt.np+ng)) then
                              call cova3(xdb(j1),ydb(j1),zdb(j1),
     +                              xx,yy,zz,ind,nst,MAXNST,c0,it,cc,aa,
     +                              gradh,angh,1,MAXROT,rotmat,cmax,cova)
```

```
                        else
                              xi1 = xx - ddxa(j)
                              xi2 = xx + ddxa(j)
                              yi1 = yy - ddya(j)
                              yi2 = yy + ddya(j)
                              call cova3(xdb(1),ydb(1),zdb(1),xi1,
     +                              yi1,zz,ind,nst,MAXNST,c0,it,cc,aa,
     +                              gradh,angh,1,MAXROT,rotmat,cmax,cov1)
                              call cova3(xdb(1),ydb(1),zdb(1),xi2,
     +                              yi2,zz,ind,nst,MAXNST,c0,it,cc,aa,
     +                              gradh,angh,1,MAXROT,rotmat,cmax,cov2)
                              cova = cov1 - cov2
                        end if
                        dx = xx - xdb(j1)
                        dy = yy - ydb(j1)
                        dz = zz - zdb(j1)
                        if((dx*dx+dy*dy+dz*dz).lt.EPSLON) then
                              cb = cb + cova - c0(ind)
                        else
                              cb = cb + cova
                        end if
                  end do
                  cb = cb / real(ndb)
            endif
            r(j)  = dble(cb)
         end if
      end do
c
c Set up for either simple or ordinary cokriging:
c
      if(ktype.eq.1) then
            do i=1,na
                  a(neq*(i-1)+na+1) = dble(unbias)
                  a(neq*na+i)       = dble(unbias)
            end do
      else if(ktype.eq.2) then
            do i=1,mdt
                  lim = na + i
                  do k=1,np
                        a(neq*(lim-1)+k) = dble(unbias)
                        a(neq*(k-1)+lim) = dble(unbias)
                  end do
                  if(whatest.eq.0) r(lim) = dble(unbias)
            end do
            do j=1,(nvr-1)
                  lim2 = na + mdt + j
                  do k=(np+ng),na
                        a(neq*(lim2-1)+k) = dble(unbias)
                        a(neq*(k-1)+lim2) = dble(unbias)
                  end do
                  if(whatest.ne.0) r(lim2) = dble(unbias)
            end do
      endif
c
c Add the additional unbiasedness constraints:
c
      if(ktype.eq.2) then
      im = na + 1
```

```
c
c First drift term (linear in "x"):
c
      if(idrif(1).eq.1) then
            im=im+1
            do k=1,np
                  a(neq*(im-1)+k) = dble(xa(k)*resc)
                  a(neq*(k-1)+im) = dble(xa(k)*resc)
            end do
            do k=(np+1),(np+ng)
                  xk1 = xa(k) - ddxa(k)
                  xk2 = xa(k) + ddxa(k)
                  a(neq*(im-1)+k) = dble((xk1-xk2)*resc)
                  a(neq*(k-1)+im) = dble((xk1-xk2)*resc)
            end do
            if(whatest.eq.0) r(im) = dble(bv(1))
      endif
c
c Second drift term (linear in "y"):
c
      if(idrif(2).eq.1) then
            im=im+1
            do k=1,np
                  a(neq*(im-1)+k) = dble(ya(k)*resc)
                  a(neq*(k-1)+im) = dble(ya(k)*resc)
            end do
            do k=(np+1),(np+ng)
                  yk1 = ya(k) - ddya(k)
                  yk2 = ya(k) + ddya(k)
                  a(neq*(im-1)+k) = dble((yk1-yk2)*resc)
                  a(neq*(k-1)+im) = dble((yk1-yk2)*resc)
            end do
            if(whatest.eq.0) r(im) = dble(bv(2))
      endif
c
c Third drift term (linear in "z"):
c
      if(idrif(3).eq.1) then
            im=im+1
            do k=1,np
                  a(neq*(im-1)+k) = dble(za(k)*resc)
                  a(neq*(k-1)+im) = dble(za(k)*resc)
            end do
            if(whatest.eq.0) r(im) = dble(bv(3))
      endif
c
c Fourth drift term (quadratic in "x"):
c
      if(idrif(4).eq.1) then
            im=im+1
            do k=1,np
                  a(neq*(im-1)+k) = dble(xa(k)*xa(k)*resc)
                  a(neq*(k-1)+im) = dble(xa(k)*xa(k)*resc)
            end do
            do k=(np+1),(np+ng)
                  xk1 = xa(k) - ddxa(k)
                  xk2 = xa(k) + ddxa(k)
                  a(neq*(im-1)+k) = dble((xk1*xk1-xk2*xk2)*resc)
```

```
                        a(neq*(k-1)+im) = dble((xk1*xk1-xk2*xk2)*resc)
                end do
                if(whatest.eq.0) r(im) = dble(bv(4))
        endif
c
c Fifth drift term (quadratic in "y"):
c
        if(idrif(5).eq.1) then
                im=im+1
                do k=1,np
                        a(neq*(im-1)+k) = dble(ya(k)*ya(k)*resc)
                        a(neq*(k-1)+im) = dble(ya(k)*ya(k)*resc)
                end do
                do k=(np+1),(np+ng)
                        yk1 = ya(k) - ddya(k)
                        yk2 = ya(k) + ddya(k)
                        a(neq*(im-1)+k) = dble((yk1*yk1-yk2*yk2)*resc)
                        a(neq*(k-1)+im) = dble((yk1*yk1-yk2*yk2)*resc)
                end do
                if(whatest.eq.0) r(im) = dble(bv(5))
        endif
c
c Sixth drift term (quadratic in "z"):
c
        if(idrif(6).eq.1) then
                im=im+1
                do k=1,np
                        a(neq*(im-1)+k) = dble(za(k)*za(k)*resc)
                        a(neq*(k-1)+im) = dble(za(k)*za(k)*resc)
                end do
                if(whatest.eq.0) r(im) = dble(bv(6))
        endif
c
c Seventh drift term (quadratic in "xy"):
c
        if(idrif(7).eq.1) then
                im=im+1
                do k=1,np
                        a(neq*(im-1)+k) = dble(xa(k)*ya(k)*resc)
                        a(neq*(k-1)+im) = dble(xa(k)*ya(k)*resc)
                end do
                do k=(np+1),(np+ng)
                        xk1 = xa(k) - ddxa(k)
                        xk2 = xa(k) + ddxa(k)
                        yk1 = ya(k) - ddya(k)
                        yk2 = ya(k) + ddya(k)
                        a(neq*(im-1)+k) = dble((xk1*yk1-xk2*yk2)*resc)
                        a(neq*(k-1)+im) = dble((xk1*yk1-xk2*yk2)*resc)
                end do
                if(whatest.eq.0) r(im) = dble(bv(7))
        endif
c
c Eighth drift term (quadratic in "xz"):
c
        if(idrif(8).eq.1) then
                im=im+1
                do k=1,np
                        a(neq*(im-1)+k) = dble(xa(k)*za(k)*resc)
```

```
                a(neq*(k-1)+im) = dble(xa(k)*za(k)*resc)
            end do
            do k=(np+1),(np+ng)
                xk1 = xa(k) - ddxa(k)
                xk2 = xa(k) + ddxa(k)
                a(neq*(im-1)+k) = dble((xk1-xk2)*za(k)*resc)
                a(neq*(k-1)+im) = dble((xk1-xk2)*za(k)*resc)
            end do
            if(whatest.eq.0) r(im) = dble(bv(8))
      endif
c
c Ninth drift term (quadratic in "yz"):
c
      if(idrif(9).eq.1) then
            im=im+1
            do k=1,np
                a(neq*(im-1)+k) = dble(ya(k)*za(k)*resc)
                a(neq*(k-1)+im) = dble(ya(k)*za(k)*resc)
            end do
            do k=(np+1),(np+ng)
                yk1 = ya(k) - ddya(k)
                yk2 = ya(k) + ddya(k)
                a(neq*(im-1)+k) = dble((yk1-yk2)*za(k)*resc)
                a(neq*(k-1)+im) = dble((yk1-yk2)*za(k)*resc)
            end do
            if(whatest.eq.0) r(im) = dble(bv(9))
      endif
      end if
c
c Copy the right hand side to compute the kriging variance later:
c
      do k=1,neq
            rr(k) = r(k)
      end do
c
c Write out the kriging Matrix if Seriously Debugging:
c
      if(idbg.ge.3) then
            write(ldbg,*) '         '
            write(ldbg,*) '****************************************'
            write(ldbg,*) '         '
            write(ldbg,*) 'Estimating node index : ',ix,iy,iz
            is = 1 - neq
            do i=1,neq
                is = 1 + (i-1)*neq
                ie = is + neq - 1
                write(ldbg,103) i,r(i),(a(j),j=is,ie)
 103            format('     r(',i3,') =',f7.4,'   a= ',9(10f7.4))
            end do
      endif
c
c Solve the kriging system:
c
      call ktsol(neq,1,1,a,r,s,ising,MAXEQ)
c
c Write a warning if the matrix is singular:
c
      if(ising.ne.0) then
```

```
                write(ldbg,*) 'WARNING COKTBC: singular matrix'
                write(ldbg,*) '         for block',ix,iy,iz
                write(lout,100) UNEST,UNEST
                go to 4
            endif
c
c Write the kriging weights and data if requested:
c
            if(idbg.ge.2) then
                write(ldbg,*) '         '
                write(ldbg,*) 'BLOCK: ',ix,iy,iz,' at ',xloc,yloc,zloc
                write(117,'(2f12.2)') xloc,yloc
                write(ldbg,*) ' '
                if(ktype.eq.1) then
                    write(ldbg,*) '  Lagrange multiplier: ',s(na+1)
                else if(ktype.ge.2) then
                    write(ldbg,*) '  Lagrange multiplier: ',s(na+1)
                    write(ldbg,*) '  Lagrange multiplier: ',s(na+mdt+1)
                endif
                write(ldbg,*) '  np, ng, ns and na : ', np, ng, ns, na
                write(ldbg,*) '  BLOCK EST: x, y, z, vr, ddx, ddy, wt '
                do i=1,na
                write(ldbg,'(6f9.2,f12.3)') xa(i),ya(i),za(i),vra(i),
     +                                      ddxa(i),ddya(i),s(i)
                end do
            endif
c
c Compute the estimate and the kriging variance:
c
            sumw = 0.0
            ook  = 0.0
            ookv = cbb
            do i=1,neq
                if(i.le.na) then
                    ookv = ookv - real(s(i))*rr(i)
                    sumw = sumw + real(s(i))
                    ook  = ook  + real(s(i))*vra(i)
                else
                    ookv = ookv - real(s(i))*rr(i)
                endif
            end do
c
c Add mean if SK:
c
            ook = ook + (1.0-sumw)*vmean(1)
c
c Write results:
c
            ncells = ncells + 1
c           if ((whatest.eq.0).and.(logopt1.eq.1)) ook = exp(ook + 0.5*ookv -
c     +                                                      s(na+1))
c           if ((whatest.eq.1).and.(logopt2.eq.1)) ook = exp(ook + 0.5*ookv -
c     +                                                      s(na+mdt+1))
            if ((whatest.eq.0).and.(logopt1.eq.1)) ook = ook / log(10.)
            if ((whatest.eq.1).and.(logopt2.eq.1)) ook = ook / log(10.)
            if (ook.gt.restmax.and.nrestmax.ne.0) ook = restmax
            if (ook.lt.restmin.and.nrestmin.ne.0) ook = restmin
            krigout(ncells) = ook
```

```
        if (ook.gt.datamax) datamax = ook
        if (ook.lt.datamin) datamin = ook
        if (whatest.eq.0) then
              write(lout,100) ook,ookv
        else
              write(lout,1000) ook,ookv
        endif
 100  format(f12.4,1x,f12.4)
 1000 format(f21.13,1x,f12.4)
c
c Accumulate statistics of kriged blocks:
c
      nk = nk + 1
      uk = uk + ook
      vk = vk + ook*ook
      if(idbg.ge.3) write(ldbg,*) ' estimate, variance  ',ook,ookv
c
c END OF MAIN LOOP OVER ALL THE BLOCKS:
c
 4    continue
c
c I/O Files Format Issues
c
      if(noutfile.eq.1) then
              write(21,*) datamin,datamax
              if (datamin.lt.0.01) then
                   write(21,322)(krigout(iii),iii=1,ncells)
            else
                   write(21,323)(krigout(iii),iii=1,ncells)
            endif
      else if(noutfile.eq.2) then
              write(21) dble(datamin),dble(datamax)
            write(21)(krigout(iii),iii=1,ncells)
      else if(noutfile.eq.3) then
            write(21) dble(datamin),dble(datamax)
              rottemp = 0.0
            blnkval = 1.70141e38
            dataid = 1096040772
            datalen = nx*ny*8
              write(21)rottemp,blnkval,dataid,datalen
            write(21)(dble(krigout(iii)),iii=1,ncells)
      end if
 322  format(<NX>e18.7)
 323  format(<NX>f18.5)
        deallocate (krigout)
c
c Write statistics of kriged values:
c
      if(nk.gt.0.and.idbg.gt.0) then
            vk = (vk-uk*uk/real(nk))/real(nk)
            uk = uk/real(nk)
            write(ldbg,*)
            write(ldbg,*) 'Estimated  ',nk,' blocks '
            write(ldbg,*) '  average  ',uk
            write(ldbg,*) '  variance ',vk
            write(*,*)
            write(*,*)    'Estimated  ',nk,' blocks '
            write(*,*)    '  average  ',uk
```

```
            write(*,*)      '  variance ',vk
      endif
      return
      end


      subroutine makepar
c-----------------------------------------------------------------------
c
c                      Write a Parameter File
c                      **********************
c
c
c
c-----------------------------------------------------------------------
      lun = 99
      open(lun,file='coktbc.par',status='UNKNOWN')
      write(lun,10)
 10   format('                  Parameters for COKTBC',/,
     +       '                  **********************',/,/,
     +       'START OF PARAMETERS:')

      write(lun,11)
 11   format('somedata.dat                          ',
     +       '-file with data')
      write(lun,12)
 12   format('2                                     ',
     +       '-   number of variables primary+other')
      write(lun,122)
 122   format('0                                     ',
     +       '-   estimated variable : 0=head, 1=transmittivity')
      write(lun,13)
 13   format('1   2   0   3   4   5                 ',
     +       '-   columns for X,Y,Z and variables')
      write(lun,14)
 14   format('-9e+6     1e+29                       ',
     +       '-   trimming limits')
      write(lun,15)
 15   format('0                                     ',
     +       '-co-located cokriging? (0=no, 1=yes)')
      write(lun,16)
 16   format('somedata.dat                          ',
     +       '-   file with gridded covariate')
      write(lun,17)
 17   format('4                                     ',
     +       '-   column for covariate')
      write(lun,18)
 18   format('3                                     ',
     +       '-debugging level: 0,1,2,3')
      write(lun,19)
 19   format('coktbc.dbg                            ',
     +       '-file for debugging output')
      write(lun,20)
 20   format('coktbc.out                            ',
     +       '-file for output')
      write(lun,21)
 21   format('51   0.0   10.0                       ',
```

```
     +       '-nx,xmn,xsiz')
             write(lun,22)
22           format('51    0.0    10.0                    ',
     +       '-ny,ymn,ysiz')
             write(lun,23)
23           format('1     0.5    1.0                     ',
     +       '-nz,zmn,zsiz')
             write(lun,24)
24           format('1     1      1                       ',
     +       '-x, y, and z block discretization')
             write(lun,241)
241          format('0                                    ',
     +       '-0=constant or 1=moving neighborhood')
             write(lun,25)
25           format('1    50    50    50                  ',
     +       '-min primary,max primary,max all sec')
             write(lun,26)
26           format('710.0  710.0  0.0                    ',
     +       '-maximum search radii: primary')
             write(lun,27)
27           format('710.0  710.0  0.0                    ',
     +       '-maximum search radii: all secondary')
             write(lun,28)
28           format(' 0.0    0.0    0.0                   ',
     +       '-angles for search ellipsoid')
             write(lun,29)
29           format('2                                    ',
     +       '-kriging type (0=SK, 1=OK, 2=OK-trad)')
             write(lun,291)
291          format('1 1 0 0 0 0 0 0 0                    ',
     +       '-drift: x,y,z,xx,yy,zz,xy,xz,zy')
             write(lun,30)
30           format('0.00   0.00   0.00   0.00            ',
     +       '-mean(i),i=1,nvar')
             write(lun,301)
301          format('1                                    ',
     +       '-mean of transmissivity')
             write(lun,302)
302          format('0.1    270.0                         ',
     +       '-head gradient, gradient angle with X axis')
             write(lun,303)
303          format('1                                    ',
     +       '-grid file type')
             write(lun,304)
304          format('0      1                             ',
     +       '-use_log interpolation flag')
             write(lun,305)
305          format('0    0                               ',
     +       '-Restrict_Min_Value, Value_Min')
             write(lun,306)
306          format('0    0                               ',
     +       '-Restrict_Max_Value, Value_Max')
             write(lun,31)
31           format('1      1                             ',
     +       '-semivariogram for "i" and "j"')
             write(lun,32)
32           format('1    0.01                            ',
     +       '-    nst, nugget effect')
```

```
      write(lun,33)
 33   format('7    100.0  0.0    0.0    0.0          ',
     +       '-   it,cc,ang1,ang2,ang3')
      write(lun,34)
 34   format('         710.0  710.0  1.0            ',
     +       '-   a_hmax, a_hmin, a_vert')
      write(lun,35)
 35   format('1      2                               ',
     +       '-semivariogram for "i" and "j"')
      write(lun,36)
 36   format('1     0.0                              ',
     +       '-   nst, nugget effect')
      write(lun,37)
 37   format('8    1.0  0.0    0.0    0.0            ',
     +       '-   it,cc,ang1,ang2,ang3')
      write(lun,38)
 38   format('         710.0  710.0  1.0            ',
     +       '-   a_hmax, a_hmin, a_vert')
      write(lun,43)
 43   format('2      2                               ',
     +       '-semivariogram for "i" and "j"')
      write(lun,44)
 44   format('1     10.0                             ',
     +       '-   nst, nugget effect')
      write(lun,45)
 45   format('1    1.0  0.0    0.0    0.0            ',
     +       '-   it,cc,ang1,ang2,ang3')
      write(lun,46)
 46   format('         710.0  710.0  1.0            ',
     +       '-   a_hmax, a_hmin, a_vert')

      close(lun)
      return
      end
```

Appendix B

GSLIB Code : Other subroutines

B.1 Include file COKTBC.inc

```
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C                                                                            %
C Copyright (C) 1996, The Board of Trustees of the Leland Stanford           %
C Junior University.  All rights reserved.                                   %
C                                                                            %
C The programs in GSLIB are distributed in the hope that they will be       %
C useful, but WITHOUT ANY WARRANTY.  No author or distributor accepts       %
C responsibility to anyone for the consequences of using them or for        %
C whether they serve any particular purpose or work at all, unless he       %
C says so in writing.  Everyone is granted permission to copy, modify       %
C and redistribute the programs in GSLIB, but only under the condition      %
C that this notice and the above copyright notice remain intact.            %
C                                                                            %
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c-----------------------------------------------------------------------
c
c               Ordinary CoKriging of a 3-D Rectangular Grid
c               ********************************************
c
c The following Parameters control static dimensioning within coktbc:
c
c   MAXSBX    maximum super block nodes in X
c   MAXSBY    maximum super block nodes in Y
c   MAXSBZ    maximum super block nodes in Z
c   MAXDAT    maximum number of data points
c   MAXVAR    maximum number of variables (including primary!)
c   MAXSAM    maximum number of data points
c   MAXCOK    maximum number of data points in kriging system
c   MAXDIS    maximum number of discretization points per block
c   MAXNST    maximum number of nested structures
c
c-----------------------------------------------------------------------
c
c User Adjustable Parameters:
c
      parameter(MAXSBX =     21, MAXSBY =  21, MAXSBZ =    11,
     +          MAXDAT =250000, MAXVAR =   2, MAXSAM =  1000,
     +          MAXDIS =     64, MAXNST =   4, MAXDT  =     9)
```

```
c
c Fixed Parameters:
c
      parameter(UNEST=-999.0,EPSLON=0.000001,MXVARG=MAXVAR*MAXVAR,
     +          MAXCOK=(MAXSAM*MAXVAR),MAXROT=MAXNST+1,
     +          MAXSB=MAXSBX*MAXSBY*MAXSBZ,
     +          MAXEQ=(MAXSAM*MAXVAR+MAXVAR+MAXDT),VERSION=1.000)
c
c Static Array Dimensioning:
c
      integer nst(MXVARG),it(MXVARG*MAXNST),iva(MAXCOK),nisb(MAXSB),
     +        ixsbtosr(8*MAXSB),iysbtosr(8*MAXSB),izsbtosr(8*MAXSB),
     +        idrif(MAXDT),kod(MAXDAT)
      real    x(MAXDAT),y(MAXDAT),z(MAXDAT),vr(MAXDAT),sec1(MAXDAT),
     +        ddx(MAXDAT),ddy(MAXDAT),tmp(MAXDAT),close(MAXDAT),
     +        close2(MAXDAT),close3(MAXDAT),bv(9),xa(MAXCOK),
     +        ya(MAXCOK),za(MAXCOK),vra(MAXCOK),ddxa(MAXCOK),
     +        ddya(MAXCOK),xdb(MAXDIS),ydb(MAXDIS),zdb(MAXDIS),
     +        vmean(MAXVAR),c0(MXVARG),cc(MXVARG*MAXNST),
     +        aa(MXVARG*MAXNST),ang1(MXVARG*MAXNST),ang2(MXVARG*MAXNST),
     +        ang3(MXVARG*MAXNST),anis1(MXVARG*MAXNST),
     +        anis2(MXVARG*MAXNST),xs1(MAXDAT),ys1(MAXDAT),xs2(MAXDAT),
     +        ys2(MAXDAT)
      real*8  r(MAXEQ),rr(MAXEQ),s(MAXEQ),a(MAXEQ*MAXEQ),unbias,
     +        rotmat(MAXROT,3,3)
c
c The data and other input variables:
c
      common /datcom/ nd,x,y,z,vr,sec1,kod,ddx,ddy,ktype,nvr,whatest,
     +                vmean,tmin,tmax,nx,ny,nz,xmn,ymn,zmn,xsiz,ysiz,
     +                zsiz,idbg,ldbg,lout,newnd,nsc,xs1,ys1,xs2,ys2,
     +                csiz,noutfile,logopt1,logopt2,nrestmin,
     +                nrestmax,restmin,restmax,Tmean
c
c Kriging parameters:
c
      common /krigcm/ ndmin,ndmaxp,ndmaxg,ndmaxs,radiusp,radiuss,noct,
     +                nxdis,nydis,nzdis,idrif,nborhood
c
c Variogram Parameters:
c
      common /vargdt/ nst,it,c0,cc,aa,ang1,ang2,ang3,anis1,anis2,
     +                gradh,angh
c
c Search variables and data for kriging:
c
      common /srccom/ sang1,sang2,sang3,sanisp1,sanisp2,isrot,saniss1,
     +                saniss2,radsqdp,radsqds,na,np,ng,ns,xa,ya,za,vra,
     +                ddxa,ddya,iva,xas,yas,zas,vras,xdb,ydb,zdb,ndb,bv
c
c Kriging systems (double precision arrays):
c
      common /krgsys/ r,rr,s,a,unbias,rotmat
```

B.2 Subroutine bdarr

```
      subroutine bdarr(n,x,y,kod,csiz,newn)
```

```
! Computes the number of points to add to determine the size of the
! data array.
          integer n, addpts, newn, i
          integer kod(n)
          real x(n),y(n)
          real csiz, dist
          addpts = 0
          newn = n
      do 111, i = 1,n-1
        if ((kod(i).ne.0).and.(kod(i).eq.kod(i+1))) then
              dist    = sqrt((x(i)-x(i+1))**2+(y(i)-y(i+1))**2)
              if (dist.lt.(2.*csiz)) then
                addpts = 1
              else if (dist.gt.(20.*csiz)) then
                addpts = 19
              else if (mod(dist,csiz).ne.0) then
                addpts = int(dist/csiz)
              else
                addpts = int(dist/csiz)-1
              endif
          newn = newn + addpts
        endif
 111  continue
          return
          end
```

B.3 Subroutine bdpts

```
      subroutine bdpts(n,x,y,z,vr,ve,kod,csiz)
! Detects the boundary points and add other points along the boundary
! lines to smoother the interpolation. Essential for Constraint Flux.
! 2D subroutine.
          integer n, addpts, i, j, k
          integer kod(n)
          real csiz, dist, slope, spacin, step, stepz, stepvr, stepve
          real x(n),y(n),z(n),vr(n),ve(n)
          addpts = 0
          i = 1
 112  if ((kod(i).ne.0).and.(kod(i).eq.kod(i+1))) then
            dist    = sqrt((x(i)-x(i+1))**2+(y(i)-y(i+1))**2)
            if (dist.lt.(2.*csiz)) then
              addpts = 1
              spacin = dist / 2.
            else if (dist.gt.(20.*csiz)) then
              addpts = 19
              spacin = dist / 20.
            else if (mod(dist,csiz).ne.0) then
              addpts = int(dist/csiz)
              spacin = csiz
            else
              addpts = int(dist/csiz)-1
              spacin = csiz
            endif
            stepz  = spacin/dist*(z(i+1)-z(i))
            stepvr = spacin/dist*(vr(i+1)-vr(i))
            stepve = spacin/dist*(ve(i+1)-ve(i))
            if (x(i).ne.x(i+1)) then
```

```
              slope  = (y(i+1)-y(i))/(x(i+1)-x(i))
              step   = spacin/sqrt(1+slope**2)
            else
              step   = 0
            endif
            do 113, k = n,i+1+addpts,-1
              x(k)   = x(k-addpts)
              y(k)   = y(k-addpts)
              z(k)   = z(k-addpts)
              vr(k)  = vr(k-addpts)
              ve(k)  = ve(k-addpts)
              kod(k) = kod(k-addpts)
113       continue
            do 114, j = 1,addpts
              if (x(i).lt.x(i+1+addpts)) then
                x(i+j) = x(i) + j*step
              else
                x(i+j) = x(i) - j*step
              endif
              if (y(i).lt.y(i+1+addpts)) then
                if (x(i).ne.x(i+1+addpts)) then
                  y(i+j) = y(i) + j*step*slope
                else
                  y(i+j) = y(i) + j*spacin
                endif
              else
                if (x(i).ne.x(i+1+addpts)) then
                  y(i+j) = y(i) - j*step*slope
                else
                  y(i+j) = y(i) - j*spacin
                endif
              endif
              z(i+j)   = z(i)  + j*stepz
              vr(i+j)  = vr(i) + j*stepvr
              ve(i+j)  = ve(i) + j*stepve
              kod(i+j) = kod(i)
114       continue
            i = i + 1 + addpts
          else
            i = i + 1
          endif
          if (i.ge.n) then
            go to 115
              endif
              go to 112
115       continue
              return
              end
```

B.4 Subroutine CCW

```
          integer function CCW(x1,y1,x2,y2,x3,y3)
! Check if the 3 points are in Counter Clock Wise order.
              real x1,y1,x2,y2,x3,y3
          if (((y3-y1)*(x2-x1)).gt.((y2-y1)*(x3-x1))) then
             CCW = 1
          else if (((y3-y1)*(x2-x1)).eq.((y2-y1)*(x3-x1))) then
```

```
      CCW = 0
   else
      CCW = -1
   endif
       return
       end
```

B.5 Subroutine CHKNAM

```
      subroutine chknam(str,len)
c-----------------------------------------------------------------------
c
c                    Check for a Valid File Name
c                    ***************************
c
c This subroutine takes the character string "str" of length "len" and
c removes all leading blanks and blanks out all characters after the
c first blank found in the string (leading blanks are removed first).
c
c
c
c-----------------------------------------------------------------------
      parameter (MAXLEN=132)
      character str(MAXLEN)*1
c
c Remove leading blanks:
c
      do i=1,len-1
            if(str(i).ne.' ') then
                  if(i.eq.1) go to 1
                  do j=1,len-i+1
                        k = j + i - 1
                        str(j) = str(k)
                  end do
                  do j=len,len-i+2,-1
                        str(j) = ' '
                  end do
                  go to 1
            end if
      end do
 1    continue
c
c Find first blank and blank out the remaining characters:
c
      do i=1,len-1
            if(str(i).eq.' ') then
                  do j=i+1,len
                        str(j) = ' '
                  end do
                  go to 2
            end if
      end do
 2    continue
c
c Return with modified file name:
c
      return
```

```
      end
```

B.6 Subroutine COVA3

```
      subroutine cova3(x1,y1,z1,x2,y2,z2,ivarg,nst,MAXNST,c0,it,cc,aa,
     +                 gradh,angh,irot,MAXROT,rotmat,cmax,cova)
c-----------------------------------------------------------------------
c
c                     Covariance Between Two Points
c                     *****************************
c
c This subroutine calculated the covariance associated with a variogram
c model specified by a nugget effect and nested varigoram structures.
c The anisotropy definition can be different for each nested structure.
c
c
c
c INPUT VARIABLES:
c
c   x1,y1,z1           coordinates of first point
c   x2,y2,z2           coordinates of second point
c   nst(ivarg)         number of nested structures (maximum of 4)
c   ivarg              variogram number (set to 1 unless doing cokriging
c                        or indicator kriging)
c   MAXNST             size of variogram parameter arrays
c   c0(ivarg)          isotropic nugget constant
c   it(i)              type of each nested structure:
c                        1. spherical model of range a;
c                        2. exponential model of parameter a;
c                             i.e. practical range is 3a
c                        3. gaussian model of parameter a;
c                             i.e. practical range is a*sqrt(3)
c                        4. power model of power a (a must be gt. 0  and
c                             lt. 2).  if linear model, a=1,c=slope.
c                        5. hole effect model
c   cc(i)              multiplicative factor of each nested structure.
c                        (sill-c0) for spherical, exponential,and gaussian
c                        slope for linear model.
c   aa(i)              parameter "a" of each nested structure.
c   gradh              value of the hydraulic head gradient (for it = 7)
c   angh               angle between the direction of the hydraulic head
c                        gradient and the X axis (for it = 7)
c   irot               index of the rotation matrix for the first nested
c                      structure (the second nested structure will use
c                      irot+1, the third irot+2, and so on)
c   MAXROT             size of rotation matrix arrays
c   rotmat             rotation matrices
c
c
c OUTPUT VARIABLES:
c
c   cmax               maximum covariance
c   cova               covariance between (x1,y1,z1) and (x2,y2,z2)
c
c
c
c EXTERNAL REFERENCES: sqdist      computes anisotropic squared distance
```

```
c                     rotmat    computes rotation matrix for distance
c-----------------------------------------------------------------------
      parameter(PI=3.14159265,PMX=999.,EPSLON=1.e-10)
      integer   nst(*),it(*)
      real      c0(*),cc(*),aa(*),gradh,angh,hr,dxa,ct,h1,hr1
      real*8    rotmat(MAXROT,3,3),hsqd,sqdist
c
c Calculate the maximum covariance value (used for zero distances and
c for power model covariance):
c
      istart = 1 + (ivarg-1)*MAXNST
      cmax   = c0(ivarg)
      do is=1,nst(ivarg)
            ist = istart + is - 1
            if(it(ist).eq.4) then
                  cmax = cmax + PMX
            else if(it(ist).eq.7) then
                  ct = cc(ist) * (gradh * aa(ist) / 4.)**2
                  cmax = cmax + 20.*ct
            else if(it(ist).eq.8) then
                  cmax = cc(ist)*gradh*aa(ist)*2
            else
                  cmax = cmax + cc(ist)
            endif
      end do
c
c Check for "zero" distance, return with cmax if so:
c
      hsqd = sqdist(x1,y1,z1,x2,y2,z2,irot,MAXROT,rotmat)
      if(real(hsqd).lt.EPSLON) then
            cova = cmax
            return
      endif
c
c Loop over all the structures:
c
      cova = 0.0
      do is=1,nst(ivarg)
            ist = istart + is - 1
c
c Compute the appropriate distance:
c
            if(ist.ne.1) then
                  ir = min((irot+is-1),MAXROT)
                  hsqd=sqdist(x1,y1,z1,x2,y2,z2,ir,MAXROT,rotmat)
            end if
            h = real(dsqrt(hsqd))
c
c Spherical Variogram Model?
c
            if(it(ist).eq.1) then
                  hr = h/aa(ist)
                  if(hr.lt.1.) cova=cova+cc(ist)*(1.-hr*(1.5-.5*hr*hr))
c
c Exponential Variogram Model?
c
            else if(it(ist).eq.2) then
                  cova = cova + cc(ist)*exp(-3.0*h/aa(ist))
```

```
c
c Gaussian Variogram Model?
c
            else if(it(ist).eq.3) then
                  cova = cova + cc(ist)*exp(-(3.0*h/aa(ist))
     +                                      *(3.0*h/aa(ist)))
c
c Power Variogram Model?
c
            else if(it(ist).eq.4) then
                  cova = cova + cmax - cc(ist)*(h**aa(ist))
c
c Hole Effect Model?
c
            else if(it(ist).eq.5) then
c                 d = 10.0 * aa(ist)
c                 cova = cova + cc(ist)*exp(-3.0*h/d)*cos(h/aa(ist)*PI)
                  cova = cova + cc(ist)*cos(h/aa(ist)*PI)
c
c Cubic Variogram Model?
c
            else if(it(ist).eq.6) then
                  hr = h/aa(ist)
                  if(hr.lt.1.) then
                     cova = cova + cc(ist)*(1. - 7.*hr**2
     +                    + 35./4.*hr**3 - 7./2.*hr**5 + 3./4.*hr**7)
                  endif
c
c Transmissivity Linked Spherical Variogram Model?
c
            else if(it(ist).eq.7) then
                  hr = h/aa(ist)
                  h1 = COS(2.*PI/360.*angh)*(x1-x2)
     +                 + SIN(2.*PI/360.*angh)*(y1-y2)
                  dxa = h1/aa(ist)
                  ct = cc(ist) * (gradh * aa(ist) / 4.)**2
                  if(hr.le.1.) then
                    cova = cova + cmax - ct*(hr**2 -8./15.*hr**3
     +                + 8./175.*hr**5 +(2. -8./5.*hr +8./35.*hr**3)
     +                * dxa**2)
                  else
                    cova = cova + cmax - ct*(32./75. +3./(35.*hr**2)
     +                +4./5.*LOG(hr) +(4./(5.*hr**2) -6./(35.*hr**4))
     +                * dxa**2)
                  endif
c
c Log(T)-h Cross-variogram Model?
c
            else if(it(ist).eq.8) then
                  h1 = COS(2.*PI/360.*angh)*(x1-x2)
     +                 + SIN(2.*PI/360.*angh)*(y1-y2)
                  hr  = h  / aa(ist)
                  if(hr.lt.1.) then
                     cova = cova + cmax - gradh*cc(ist)*h1*(0.5 -0.5*hr
     +                    +0.1*hr**3)
                  else
                     cova = cova + cmax - gradh*cc(ist)*h1/(10. * hr**2)
                  endif
```

```
            endif
        end do
c
c Finished:
c
        return
        end
```

B.7 Subroutine ficcoord

```
        subroutine ficcoord(n,x,y,kod,csiz,ddx,ddy)
! Create 2 fictive x & y coordinates for each prescribed flux BC point
! to improve the interpolation 2D subroutine.
        integer n, nx, ny, i, j
            integer kod(n)
            real xsiz, ysiz, xbig, ybig, perp, spacin, step
            real x(n), y(n), ddx(n), ddy(n)
            spacin = csiz
            ddx = 0.0
            ddy = 0.0
            do 116, i = 2,n-1
              if ((kod(i).lt.0).and.(kod(i-1).eq.kod(i)).and.
                  (kod(i).eq.kod(i+1))) then
                if (y(i+1).ne.y(i-1)) then
                  perp  = -(x(i+1)-x(i-1))/(y(i+1)-y(i-1))
                  step  = spacin/sqrt(1+perp**2)
                  ddx(i) = step
                  ddy(i) = step*perp
                else
                  ddy(i) = spacin
                endif
              else if ((kod(i).lt.0).and.(kod(i-1).ne.kod(i)).and.
                       (kod(i).eq.kod(i+1))) then
                if (y(i+1).ne.y(i)) then
                  perp  = -(x(i+1)-x(i))/(y(i+1)-y(i))
                  step  = spacin/sqrt(1+perp**2)
                  ddx(i) = step
                  ddy(i) = step*perp
                else
                  ddy(i) = spacin
                endif
              else if ((kod(i).lt.0).and.(kod(i-1).eq.kod(i)).and.
                       (kod(i).ne.kod(i+1))) then
                if (y(i).ne.y(i-1)) then
                  perp  = -(x(i)-x(i-1))/(y(i)-y(i-1))
                  step  = spacin/sqrt(1+perp**2)
                  ddx(i) = step
                  ddy(i) = step*perp
                else
                  ddy(i) = spacin
                endif
              endif
            if ((kod(1).lt.0).and.(kod(2).eq.kod(1))) then
              if (y(2).ne.y(1)) then
                perp  = -(x(2)-x(1))/(y(2)-y(1))
                step  = spacin/sqrt(1+perp**2)
                ddx(1) = step
```

```
            ddy(1) = step*perp
          else
            ddy(1) = spacin
          endif
        endif
        if ((kod(n).lt.0).and.(kod(n-1).eq.kod(n))) then
          if (y(n).ne.y(n-1)) then
            perp   = -(x(n)-x(n-1))/(y(n)-y(n-1))
            step   = spacin/sqrt(1+perp**2)
            ddx(n) = step
            ddy(n) = step*perp
          else
            ddy(n) = spacin
          endif
        endif
 116  continue
        return
        end
```

B.8 Subroutine fluxcoor

```
      subroutine fluxcorr(n,x,y,vr,kod,csiz,Tmean)
! Detects the constant flux points and apply the correction factor to
! convert the volumic flow rate to the hydraulic head difference>
        integer n, i, j, k
        integer kod(n)
        real csiz, Tmean, dist
        real x(n), y(n), vr(n)
        do 117, i=1,n-1
          do 118, j=2,n
            if(i.ne.1.and.j.ne.n) then
              if(kod(i).lt.0.and.kod(i-1).ne.kod(i).and.vr(i).ne.0.and.&
     &           kod(j).eq.kod(i).and.kod(j+1).ne.kod(j)) then
                dist   = sqrt((x(i)-x(j))**2+(y(i)-y(j))**2)
                do 119, k=i,j
                  vr(k) = vr(k)*(2*csiz)**2/(Tmean*dist)
 119          continue
            endif
          else if(i.eq.1.and.j.ne.n) then
            if(kod(1).lt.0.and.vr(1).ne.0.and.kod(j).eq.kod(1).and.&
     &         kod(j+1).ne.kod(j)) then
                dist   = sqrt((x(1)-x(j))**2+(y(1)-y(j))**2)
                do 120, k=1,j
                  vr(k) = vr(k)*(2*csiz)**2/(Tmean*dist)
 120          continue
            endif
          else if(i.ne.1.and.j.eq.n) then
            if(kod(i).lt.0.and.kod(i-1).ne.kod(i).and.vr(i).ne.0.and.&
     &         kod(n).eq.kod(i)) then
                dist   = sqrt((x(i)-x(n))**2+(y(i)-y(n))**2)
                do 121, k=i,n
                  vr(k) = vr(k)*(2*csiz)**2/(Tmean*dist)
 121          continue
            endif
          else
            if(kod(1).lt.0.and.vr(1).ne.0.and.kod(n).eq.kod(1)) then
                dist   = sqrt((x(1)-x(n))**2+(y(1)-y(n))**2)
```

```
                do 122, k=1,n
                  vr(k) = vr(k)*(2*csiz)**2/(Tmean*dist)
122           continue
            endif
          end if
118     continue
117   continue
          return
          end
```

B.9 Subroutine GETINDX

```
      subroutine getindx(n,min,siz,loc,index,inflag)
c-----------------------------------------------------------------------
c
c     Gets the coordinate index location of a point within a grid
c     ***********************************************************
c
c
c n         number of "nodes" or "cells" in this coordinate direction
c min       origin at the center of the first cell
c siz       size of the cells
c loc       location of the point being considered
c index     output index within [1,n]
c inflag    true if the location is actually in the grid (false otherwise
c           e.g., if the location is outside then index will be set to
c           nearest boundary
c
c
c
c-----------------------------------------------------------------------
      integer   n,index
      real      min,siz,loc
      logical   inflag
c
c Compute the index of "loc":
c
      index = int( (loc-min)/siz + 1.5 )
c
c Check to see if in or out:
c
      if(index.lt.1) then
            index  = 1
            inflag = .false.
      else if(index.gt.n) then
            index  = n
            inflag = .false.
      else
            inflag = .true.
      end if
c
c Return to calling program:
c
      return
      end
```

B.10 Subroutine getopenfilename

```
! Example of calling the Win32 API routine GetOpenFileName
! This can be used from any application type, including Console
! Make sure that comdlg32.lib is included in the list of libraries
! to be searched.
!
! GetSaveFileName is very similar.
!
! NOTE!  You must have DVF 5.0B or later to compile this example!
!
subroutine fileopen(fname)
use dfwin
implicit none

! Declare structure used to pass and receive attributes
!
type(T_OPENFILENAME) ofn

! Declare filter specification.  This is a concatenation of
! pairs of null-terminated strings.  The first string in each pair
! is the file type name, the second is a semicolon-separated list
! of file types for the given name.  The list ends with a trailing
! null-terminated empty string.
!
character*(*) fname
character*(*),parameter :: filter_spec = &
  "Parameter Files"C//"*.par"C// &
  "Surfer Files"C//"*.grd;*.grd"C//""C

! Declare string variable to return the file specification.
! Initialize with an initial filespec, if any - null string
! otherwise
!
character*512 :: file_spec = ""C
integer status,ilen
ofn%lStructSize = SIZEOF(ofn)
ofn%hwndOwner = NULL  ! For non-console applications,
                      ! set this to the Hwnd of the
                      ! Owner window.  For QuickWin
                      ! and Standard Graphics projects,
                      ! use GETHWNDQQ(QWIN$FRAMEWINDOW)
                      !
ofn%hInstance = NULL  ! For Win32 applications, you
                      ! can set this to the appropriate
                      ! hInstance
                      !
ofn%lpstrFilter = loc(filter_spec)
ofn%lpstrCustomFilter = NULL
ofn%nMaxCustFilter = 0
ofn%nFilterIndex = 1 ! Specifies initial filter value
ofn%lpstrFile = loc(file_spec)
ofn%nMaxFile = sizeof(file_spec)
ofn%nMaxFileTitle = 0
ofn%lpstrInitialDir = NULL  ! Use Windows default directory
ofn%lpstrTitle = loc(""C)
ofn%Flags = OFN_PATHMUSTEXIST
```

```
ofn%lpstrDefExt = loc("par"C)
ofn%lpfnHook = NULL
ofn%lpTemplateName = NULL

! Call GetOpenFileName and check status
!
status = GetOpenFileName(ofn)
if (status .eq. 0) then
  type *,'No file name specified'
else
  ! Get length of file_spec by looking for trailing NUL
  ilen = INDEX(file_spec,CHAR(0))
  type *,'Filespec is ',file_spec(1:ilen-1)
  ! Example of how to see if user said "Read Only"
  !
  if (IAND(ofn%flags,OFN_READONLY) /= 0) &
    type *,'Readonly was requested'
end if
fname(1:ilen-1) = file_spec(1:ilen-1)
end subroutine fileopen
```

B.11 Subroutine getopenfilesurf

```
! Example of calling the Win32 API routine GetOpenFileName
! This can be used from any application type, including Console
! Make sure that comdlg32.lib is included in the list of libraries
! to be searched.
!
! GetSaveFileName is very similar.
!
! NOTE!  You must have DVF 5.0B or later to compile this example!
!
subroutine fopensurf(fname)
use dfwin
implicit none

! Declare structure used to pass and receive attributes
!
type(T_OPENFILENAME) ofn

! Declare filter specification.  This is a concatenation of
! pairs of null-terminated strings.  The first string in each pair
! is the file type name, the second is a semicolon-separated list
! of file types for the given name.  The list ends with a trailing
! null-terminated empty string.
!
character*(*) fname
character*(*),parameter :: filter_spec = &
  "Surfer Files"C//"*.grd"C// &
  "Surfer Files"C//"*.grd;*.grd"C//""C

! Declare string variable to return the file specification.
! Initialize with an initial filespec, if any - null string
! otherwise
!
character*512 :: file_spec = ""C
integer status,ilen
```

```
ofn%lStructSize = SIZEOF(ofn)
ofn%hwndOwner = NULL  ! For non-console applications,
                      ! set this to the Hwnd of the
                      ! Owner window.  For QuickWin
                      ! and Standard Graphics projects,
                      ! use GETHWNDQQ(QWIN$FRAMEWINDOW)
                      !
ofn%hInstance = NULL  ! For Win32 applications, you
                      ! can set this to the appropriate
                      ! hInstance
                      !
ofn%lpstrFilter = loc(filter_spec)
ofn%lpstrCustomFilter = NULL
ofn%nMaxCustFilter = 0
ofn%nFilterIndex = 1 ! Specifies initial filter value
ofn%lpstrFile = loc(file_spec)
ofn%nMaxFile = sizeof(file_spec)
ofn%nMaxFileTitle = 0
ofn%lpstrInitialDir = NULL  ! Use Windows default directory
ofn%lpstrTitle = loc(""C)
ofn%Flags = OFN_PATHMUSTEXIST
ofn%lpstrDefExt = loc("grd"C)
ofn%lpfnHook = NULL
ofn%lpTemplateName = NULL

! Call GetOpenFileName and check status
!
status = GetOpenFileName(ofn)
if (status .eq. 0) then
  type *,'No file name specified'
else
  ! Get length of file_spec by looking for trailing NUL
  ilen = INDEX(file_spec,CHAR(0))
  type *,'Filespec is ',file_spec(1:ilen-1)
  ! Example of how to see if user said "Read Only"
  !
  if (IAND(ofn%flags,OFN_READONLY) /= 0) &
    type *,'Readonly was requested'
end if
fname(1:ilen-1) = file_spec(1:ilen-1)
end subroutine fopensurf
```

B.12 Subroutine intersect

```
      logical function intersect(x11,y11,x12,y12,x21,y21,x22,y22)
! Check if the 2 segments defined by their endpoints intersect.
          real x11,y11,x12,y12,x21,y21,x22,y22
          integer CCW
          if (CCW(x11,y11,x12,y12,x21,y21).eq.0) then
          intersect=.false.
      else if (CCW(x11,y11,x12,y12,x22,y22).eq.0) then
          intersect=.false.
      else if (CCW(x11,y11,x21,y21,x22,y22).eq.0) then
          intersect=.false.
      else if (CCW(x12,y12,x21,y21,x22,y22).eq.0) then
          intersect=.false.
      else if (CCW(x11,y11,x21,y21,x22,y22).eq.CCW(x12,y12,x21,y21, &
```

```
     &          x22,y22)) then
           intersect = .false.
       else if (CCW(x11,y11,x12,y12,x21,y21).eq.CCW(x11,y11,x12,y12, &
     &          x22,y22)) then
           intersect = .false.
       else
           intersect = .true.
       end if
           return
           end
```

B.13 Subroutine KTSOL

```
      subroutine ktsol(n,ns,nv,a,b,x,ktilt,maxeq)
c-----------------------------------------------------------------------
c
c Solution of a system of linear equations by gaussian elimination with
c partial pivoting.  Several right hand side matrices and several
c variables are allowed.
c
c
c         NOTE: All input matrices must be in double precision
c
c
c INPUT/OUTPUT VARIABLES:
c
c   n                 Number of equations
c   ns                Number of right hand side matrices
c   nv                Number of variables.
c   a(n*n*nv)         left hand side matrices versus columnwise.
c   b(n*ns*nv)        input right hand side matrices.
c   x(n*ns*nv)        solution matrices.
c   ktilt             indicator of singularity
c                       =  0  everything is ok.
c                       = -1 n.le.1
c                       =  k  a null pivot appeared at the kth iteration.
c   tol               used in test for null pivot. depends on machine
c                       precision and can also be set for the tolerance
c                       of an ill-defined kriging system.
c
c
c-----------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      real*8 x(maxeq),a(maxeq*maxeq),b(maxeq)
c
c Make sure there are equations to solve:
c
      if(n.le.1) then
            ktilt = -1
            return
      endif
c
c Initialization:
c
      tol   = 0.1e-10
      ktilt = 0
      ntn   = n*n
```

```
      nm1    = n-1
c
c Triangulation is done variable by variable:
c
      do iv=1,nv
c
c Indices of location in vectors a and b:
c
            nva = ntn*(iv-1)
            nvb = n*ns*(iv-1)
c
c Gaussian elimination with partial pivoting:
c
            do k=1,nm1
                  kp1 = k+1
c
c Indice of the diagonal element in the kth row:
c
                  kdiag = nva+(k-1)*n+k
c
c Find the pivot - interchange diagonal element/pivot:
c
                  npiv = kdiag
                  ipiv = k
                  i1   = kdiag
                  do i=kp1,n
                        i1 = i1+1
                        if(abs(a(i1)).gt.abs(a(npiv))) then
                              npiv = i1
                              ipiv = i
                        endif
                  end do
                  t         = a(npiv)
                  a(npiv)   = a(kdiag)
                  a(kdiag) = t
c
c Test for singularity:
c
                  if(abs(a(kdiag)).lt.tol) then
                        ktilt=k
                        write(*,*) 'Singular Value,kdiag,ktilt=',
     +                                  a(kdiag),kdiag,ktilt
                        return
                  endif
c
c Compute multipliers:
c
                  i1 = kdiag
                  do i=kp1,n
                        i1     = i1+1
                        a(i1) = -a(i1)/a(kdiag)
                  end do
c
c Interchange and eliminate column per column:
c
                  j1 = kdiag
                  j2 = npiv
                  do j=kp1,n
```

```
                        j1     = j1+n
                        j2     = j2+n
                        t      = a(j2)
                        a(j2) = a(j1)
                        a(j1) = t
                        i1     = j1
                        i2     = kdiag
                        do i=kp1,n
                              i1     = i1+1
                              i2     = i2+1
                              a(i1) = a(i1)+a(i2)*a(j1)
                        end do
                  end do
c
c Interchange and modify the ns right hand matrices:
c
                  i1 = nvb+ipiv
                  i2 = nvb+k
                  do i=1,ns
                        t      = b(i1)
                        b(i1) = b(i2)
                        b(i2) = t
                        j1     = i2
                        j2     = kdiag
                        do j=kp1,n
                              j1     = j1+1
                              j2     = j2+1
                              b(j1) = b(j1)+b(i2)*a(j2)
                        end do
                        i1 = i1+n
                        i2 = i2+n
                  end do
            end do
c
c Test for singularity for the last pivot:
c
            kdiag = ntn*iv
            if(abs(a(kdiag)).lt.tol) then
                  ktilt = n
                  return
            endif
      end do
c
c End of triangulation. Now, solve back variable per variable:
c
      do iv=1,nv
c
c Indices of location in vectors a and b:
c
            nva  = ntn*iv
            nvb1 = n*ns*(iv-1)+1
            nvb2 = n*ns*iv
c
c Back substitution with the ns right hand matrices:
c
            do il=1,ns
                  do k=1,nm1
                        nmk = n-k
```

```
c
c Indice of the diagonal element of the (n-k+1)th row and of
c the (n-k+1)th element of the left hand side.
c
                        kdiag = nva-(n+1)*(k-1)
                        kb    = nvb2-(il-1)*n-k+1
                        b(kb) = b(kb)/a(kdiag)
                        t     = -b(kb)
                        i1    = kb
                        i2    = kdiag
                        do i=1,nmk
                              i1    = i1-1
                              i2    = i2-1
                              b(i1) = b(i1)+a(i2)*t
                        end do
                  end do
                  kdiag = kdiag-n-1
                  kb    = kb-1
                  b(kb) = b(kb)/a(kdiag)
            end do
c
c End of back substitution:
c
      end do
c
c Restitution of the solution:
c
      itot = n*ns*nv
      do i=1,itot
            x(i) = b(i)
      end do
c
c Finished:
c
      return
      end
```

B.14 Subroutine PICKSUPR

```
      subroutine picksup(nxsup,xsizsup,nysup,ysizsup,nzsup,zsizsup,
     +                   irot,MAXROT,rotmat,radsqd,nsbtosr,ixsbtosr,
     +                   iysbtosr,izsbtosr)
c-----------------------------------------------------------------------
c
c              Establish Which Super Blocks to Search
c              **************************************
c
c This subroutine establishes which super blocks must be searched given
c that a point being estimated/simulated falls within a super block
c centered at 0,0,0.
c
c
c
c INPUT VARIABLES:
c
c   nxsup,xsizsup     Definition of the X super block grid
c   nysup,ysizsup     Definition of the Y super block grid
```

```
c    nzsup,zsizsup     Definition of the Z super block grid
c    irot              index of the rotation matrix for searching
c    MAXROT            size of rotation matrix arrays
c    rotmat            rotation matrices
c    radsqd            squared search radius
c
c
c
c OUTPUT VARIABLES:
c
c    nsbtosr           Number of super blocks to search
c    ixsbtosr          X offsets for super blocks to search
c    iysbtosr          Y offsets for super blocks to search
c    izsbtosr          Z offsets for super blocks to search
c
c
c
c EXTERNAL REFERENCES:
c
c    sqdist            Computes anisotropic squared distance
c
c
c
c-----------------------------------------------------------------------
      real*8  rotmat(MAXROT,3,3),hsqd,sqdist,shortest
      integer ixsbtosr(*),iysbtosr(*),izsbtosr(*)
c
c MAIN Loop over all possible super blocks:
c
      nsbtosr = 0
      do i=-(nxsup-1),(nxsup-1)
      do j=-(nysup-1),(nysup-1)
      do k=-(nzsup-1),(nzsup-1)
            xo = real(i)*xsizsup
            yo = real(j)*ysizsup
            zo = real(k)*zsizsup
c
c Find the closest distance between the corners of the super blocks:
c
            shortest = 1.0e21
            do i1=-1,1
            do j1=-1,1
            do k1=-1,1
                  do i2=-1,1
                  do j2=-1,1
                  do k2=-1,1
                        if(i1.ne.0.and.j1.ne.0.and.k1.ne.0.and.
     +                     i2.ne.0.and.j2.ne.0.and.k2.ne.0) then
                           xdis = real(i1-i2)*0.5*xsizsup + xo
                           ydis = real(j1-j2)*0.5*ysizsup + yo
                           zdis = real(k1-k2)*0.5*zsizsup + zo
                           hsqd = sqdist(0.0,0.0,0.0,xdis,ydis,zdis,
     +                                   irot,MAXROT,rotmat)
                           if(hsqd.lt.shortest) shortest = hsqd
                        end if
                  end do
                  end do
                  end do
```

```
                end do
                end do
                end do
c
c Keep this super block if it is close enoutgh:
c
                if(real(shortest).le.radsqd) then
                      nsbtosr = nsbtosr + 1
                      ixsbtosr(nsbtosr) = i
                      iysbtosr(nsbtosr) = j
                      izsbtosr(nsbtosr) = k
                end if
          end do
          end do
          end do
c
c Finished:
c
          return
          end
```

B.15 Subroutine remdup

```
          subroutine remdup(n,x,y,z,vr,ve,kod,ddx,ddy,npts)
!    Removes duplicate x,y pairs
          real x(n),y(n),z(n),vr(n),ve(n),kod(n),ddx(n),ddy(n)
!         write(*,*) 'n =',n
          npts = n
      do 200 i = 1,n-1
            do 300 j = i+1,n
              if (x(j).eq.x(i).and.y(j).eq.y(i)) then
                if((kod(i).ge.0.and.kod(j).ge.0).or.
                   (kod(i).le.0.and.kod(j).le.0)) then
                  do 500 k = j,n-1
                    x(k)   = x(k+1)
                    y(k)   = y(k+1)
                    z(k)   = z(k+1)
                    vr(k)  = vr(k+1)
                    ve(k)  = ve(k+1)
                    kod(k) = kod(k+1)
                    ddx(k) = ddx(k+1)
                    ddy(k) = ddy(k+1)
 500              continue
                  n = n - 1
                endif
              end if
 300      continue
 200      continue
      do 401 s = 1,n
        do 201 i = 1,n-1
              do 301 j = i+1,n
                xs1 = x(s) - ddx(s)
                xs2 = x(s) + ddx(s)
                ys1 = y(s) - ddy(s)
                ys2 = y(s) + ddy(s)
                if (x(i).eq.xs1.and.y(i).eq.ys1.and.x(j).eq.xs2.and.
                    y(j).eq.ys2) then
```

```
                do 501 k = s,n-1
                  x(k)   = x(k+1)
                  y(k)   = y(k+1)
                  z(k)   = z(k+1)
                  vr(k)  = vr(k+1)
                  ve(k)  = ve(k+1)
                  kod(k) = kod(k+1)
                  ddx(k) = ddx(k+1)
                  ddy(k) = ddy(k+1)
 501            continue
                n = n - 1
              endif
 301        continue
 201        continue
 401        continue
        ntemp = npts
    npts = n
        n = ntemp
!       write(*,*) 'npts =',npts
        return
        end
```

B.16 Subroutine scrarr

```
      subroutine scrarr(n,x,y,kod,vr,nsc)
! Computes the size of the no flow segments arrays.
          integer n, nsc, i
          integer kod(n)
          real x(n), y(n), vr(n)
          nsc = 0
      do 117, i = 1,n-1
         if (kod(i).lt.0.and.kod(i).eq.kod(i+1).and.vr(i).eq.0.and. &
     &       vr(i+1).eq.0) then
              nsc = nsc + 1
         endif
 117  continue
          return
          end
```

B.17 Subroutine screens

```
      subroutine screens(n,x,y,kod,vr,nsc,xs1,ys1,xs2,ys2)
! Fill arrays to keep in memory the end points of no flow segments.
          integer n, nsc, i, j
          integer kod(n)
          real x(n), y(n), vr(n), xs1(nsc), ys1(nsc), xs2(nsc), ys2(nsc)
      j = 0
      do 118, i = 1,n-1
         if (kod(i).lt.0.and.kod(i).eq.kod(i+1).and.vr(i).eq.0.and. &
     &       vr(i+1).eq.0) then
              j = j + 1
              xs1(j) = x(i)
              ys1(j) = y(i)
              xs2(j) = x(i+1)
              ys2(j) = y(i+1)
```

```
      endif
118   continue
      return
      end
```

B.18 Subroutine SETROT

```
      subroutine setrot(ang1,ang2,ang3,anis1,anis2,ind,MAXROT,rotmat)
c-----------------------------------------------------------------------
c
c               Sets up an Anisotropic Rotation Matrix
c               **************************************
c
c Sets up the matrix to transform cartesian coordinates to coordinates
c accounting for angles and anisotropy (see manual for a detailed
c definition):
c
c
c INPUT PARAMETERS:
c
c   ang1             Azimuth angle for principal direction
c   ang2             Dip angle for principal direction
c   ang3             Third rotation angle
c   anis1            First anisotropy ratio
c   anis2            Second anisotropy ratio
c   ind              matrix indicator to initialize
c   MAXROT           maximum number of rotation matrices dimensioned
c   rotmat           rotation matrices
c
c
c NO EXTERNAL REFERENCES
c
c
c-----------------------------------------------------------------------
      parameter(DEG2RAD=3.141592654/180.0,EPSLON=1.e-20)
      real*8    rotmat(MAXROT,3,3),afac1,afac2,sina,sinb,sint,
     +          cosa,cosb,cost
c
c Converts the input angles to three angles which make more
c  mathematical sense:
c
c         alpha   angle between the major axis of anisotropy and the
c                 E-W axis. Note: Counter clockwise is positive.
c         beta    angle between major axis and the horizontal plane.
c                 (The dip of the ellipsoid measured positive down)
c         theta   Angle of rotation of minor axis about the major axis
c                 of the ellipsoid.
c
      if(ang1.ge.0.0.and.ang1.lt.270.0) then
            alpha = (90.0   - ang1) * DEG2RAD
      else
            alpha = (450.0  - ang1) * DEG2RAD
      endif
      beta  = -1.0 * ang2 * DEG2RAD
      theta =        ang3 * DEG2RAD
c
c Get the required sines and cosines:
```

```
c
      sina   = dble(sin(alpha))
      sinb   = dble(sin(beta))
      sint   = dble(sin(theta))
      cosa   = dble(cos(alpha))
      cosb   = dble(cos(beta))
      cost   = dble(cos(theta))
c
c Construct the rotation matrix in the required memory:
c
      afac1 = 1.0 / dble(max(anis1,EPSLON))
      afac2 = 1.0 / dble(max(anis2,EPSLON))
      rotmat(ind,1,1) =        (cosb * cosa)
      rotmat(ind,1,2) =        (cosb * sina)
      rotmat(ind,1,3) =       (-sinb)
      rotmat(ind,2,1) = afac1*(-cost*sina + sint*sinb*cosa)
      rotmat(ind,2,2) = afac1*(cost*cosa + sint*sinb*sina)
      rotmat(ind,2,3) = afac1*( sint * cosb)
      rotmat(ind,3,1) = afac2*(sint*sina + cost*sinb*cosa)
      rotmat(ind,3,2) = afac2*(-sint*cosa + cost*sinb*sina)
      rotmat(ind,3,3) = afac2*(cost * cosb)
c
c Return to calling program:
c
      return
      end
```

B.19 Subroutine setsupr

```
      subroutine setsupr(nx,xmn,xsiz,ny,ymn,ysiz,nz,zmn,zsiz,nd,x,y,z, &
     &                   vr,ddx,ddy,tmp,nsec,sec,MAXSBX,MAXSBY, &
     &                   MAXSBZ,nisb,nxsup,xmnsup,xsizsup,nysup,ymnsup, &
     &                   ysizsup,nzsup,zmnsup,zsizsup)
!-----------------------------------------------------------------------
!
!           Establish Super Block Search Limits and Sort Data
!           *************************************************
!
! This subroutine sets up a 3-D "super block" model and orders the data
! by super block number.  The limits of the super block is set to the
! minimum and maximum limits of the grid; data outside are assigned to
! the nearest edge block.
!
! The idea is to establish a 3-D block network that contains all the
! relevant data.  The data are then sorted by their index location in
! the search network, i.e., the index location is given after knowing
! the block index in each coordinate direction (ix,iy,iz):
!           ii = (iz-1)*nxsup*nysup + (iy-1)*nxsup + ix
! An array, the same size as the number of super blocks, is constructed
! that contains the cumulative number of data in the model.  With this
! array it is easy to quickly check what data are located near any given
! location.
!
!
!
! INPUT VARIABLES:
!
```

```
!    nx,xmn,xsiz       Definition of the X grid being considered
!    ny,ymn,ysiz       Definition of the Y grid being considered
!    nz,zmn,zsiz       Definition of the Z grid being considered
!    nd                Number of data
!    x(nd)             X coordinates of the data
!    y(nd)             Y coordinates of the data
!    z(nd)             Z coordinates of the data
!    vr(nd)            Variable at each location.
!    ddx(nd)           X difference for Kriging under BC
!    ddy(nd)           Y difference for Kriging under BC
!    tmp(nd)           Temporary storage to keep track of the super block
!                        index associated to each data (uses the same
!                        storage already allocated for the simulation)
!    nsec              Number of secondary variables to carry with vr (max=1)
!    sec(nd)           Secondary variable (if nsec = 1)
!    MAXSB[X,Y,Z]      Maximum size of super block network
!
!
!
! OUTPUT VARIABLES:
!
!    nisb()                  Array with cumulative number of data in each
!                              super block.
!    nxsup,xmnsup,xsizsup    Definition of the X super block grid
!    nysup,ymnsup,ysizsup    Definition of the Y super block grid
!    nzsup,zmnsup,zsizsup    Definition of the Z super block grid
!
!
!
! EXTERNAL REFERENCES:
!
!    sortem            Sorting routine to sort the data
!
!
!
!-----------------------------------------------------------------------
      real    x(*),y(*),z(*),vr(*),ddx(*),ddy(*),tmp(*),sec(*)
      integer nisb(*)
      logical inflag
!
! Establish the number and size of the super blocks:
!
      nxsup   = min(nx,MAXSBX)
      nysup   = min(ny,MAXSBY)
      nzsup   = min(nz,MAXSBZ)
      xsizsup = real(nx)*xsiz/real(nxsup)
      ysizsup = real(ny)*ysiz/real(nysup)
      zsizsup = real(nz)*zsiz/real(nzsup)
      xmnsup  = (xmn-0.5*xsiz)+0.5*xsizsup
      ymnsup  = (ymn-0.5*ysiz)+0.5*ysizsup
      zmnsup  = (zmn-0.5*zsiz)+0.5*zsizsup
!
! Initialize the extra super block array to zeros:
!
      do i=1,nxsup*nysup*nzsup
            nisb(i) = 0
      end do
!
```

```
!
! Loop over all the data assigning the data to a super block and
! accumulating how many data are in each super block:
!
      do i=1,nd
            call getindx(nxsup,xmnsup,xsizsup,x(i),ix,inflag)
            call getindx(nysup,ymnsup,ysizsup,y(i),iy,inflag)
            call getindx(nzsup,zmnsup,zsizsup,z(i),iz,inflag)
            ii = ix + (iy-1)*nxsup + (iz-1)*nxsup*nysup
            tmp(i)  = ii
            nisb(ii) = nisb(ii) + 1
      end do
!
! Sort the data by ascending super block number:
!
      nsort = 6 + nsec
      call sortem(1,nd,tmp,nsort,x,y,z,vr,ddx,ddy,sec)
!
! Set up array nisb with the starting address of the block data:
!
      do i=1,(nxsup*nysup*nzsup-1)
            nisb(i+1) = nisb(i) + nisb(i+1)
      end do
!
! Finished:
!
      return
      end
```

B.20 Subroutine SORTEM

```
      subroutine sortem(ib,ie,a,iperm,b,c,d,e,f,g,h)
c-----------------------------------------------------------------------
c
c                     Quickersort Subroutine
c                     **********************
c
c This is a subroutine for sorting a real array in ascending order. This
c is a Fortran translation of algorithm 271, quickersort, by R.S. Scowen
c in collected algorithms of the ACM.
c
c The method used is that of continually splitting the array into parts
c such that all elements of one part are less than all elements of the
c other, with a third part in the middle consisting of one element.  An
c element with value t is chosen arbitrarily (here we choose the middle
c element). i and j give the lower and upper limits of the segment being
c split.  After the split a value q will have been found such that
c a(q)=t and a(l)<=t<=a(m) for all i<=l<q<m<=j.  The program then
c performs operations on the two segments (i,q-1) and (q+1,j) as follows
c The smaller segment is split and the position of the larger segment is
c stored in the lt and ut arrays.  If the segment to be split contains
c two or fewer elements, it is sorted and another segment is obtained
c from the lt and ut arrays.  When no more segments remain, the array
c is completely sorted.
c
c
c INPUT PARAMETERS:
c
```

```
c   ib,ie          start and end index of the array to be sorteda
c   a              array, a portion of which has to be sorted.
c   iperm          0 no other array is permuted.
c                  1 array b is permuted according to array a
c                  2 arrays b,c are permuted.
c                  3 arrays b,c,d are permuted.
c                  4 arrays b,c,d,e are permuted.
c                  5 arrays b,c,d,e,f are permuted.
c                  6 arrays b,c,d,e,f,g are permuted.
c                  7 arrays b,c,d,e,f,g,h are permuted.
c                 >7 no other array is permuted.
c
c   b,c,d,e,f,g,h  arrays to be permuted according to array a.
c
c OUTPUT PARAMETERS:
c
c    a       = the array, a portion of which has been sorted.
c
c    b,c,d,e,f,g,h  =arrays permuted according to array a (see iperm)
c
c NO EXTERNAL ROUTINES REQUIRED:
c
c-----------------------------------------------------------------------
      dimension a(*),b(*),c(*),d(*),e(*),f(*),g(*),h(*)
c
c The dimensions for lt and ut have to be at least log (base 2) n
c
      integer   lt(64),ut(64),i,j,k,m,p,q
c
c Initialize:
c
      j     = ie
      m     = 1
      i     = ib
      iring = iperm+1
      if (iperm.gt.7) iring=1
c
c If this segment has more than two elements  we split it
c
 10   if (j-i-1) 100,90,15
c
c p is the position of an arbitrary element in the segment we choose the
c middle element. Under certain circumstances it may be advantageous
c to choose p at random.
c
 15   p    = (j+i)/2
      ta   = a(p)
      a(p) = a(i)
      go to (21,19,18,17,16,161,162,163),iring
 163      th   = h(p)
          h(p) = h(i)
 162      tg   = g(p)
          g(p) = g(i)
 161      tf   = f(p)
          f(p) = f(i)
 16       te   = e(p)
          e(p) = e(i)
 17       td   = d(p)
```

```
          d(p) = d(i)
 18       tc   = c(p)
          c(p) = c(i)
 19       tb   = b(p)
          b(p) = b(i)
 21    continue
c
c Start at the beginning of the segment, search for k such that a(k)>t
c
       q = j
       k = i
 20    k = k+1
       if(k.gt.q)      go to 60
       if(a(k).le.ta) go to 20
c
c Such an element has now been found now search for a q such that a(q)<t
c starting at the end of the segment.
c
 30    continue
       if(a(q).lt.ta) go to 40
       q = q-1
       if(q.gt.k)      go to 30
       go to 50
c
c a(q) has now been found. we interchange a(q) and a(k)
c
 40    xa   = a(k)
       a(k) = a(q)
       a(q) = xa
       go to (45,44,43,42,41,411,412,413),iring
 413      xh   = h(k)
          h(k) = h(q)
          h(q) = xh
 412      xg   = g(k)
          g(k) = g(q)
          g(q) = xg
 411      xf   = f(k)
          f(k) = f(q)
          f(q) = xf
 41       xe   = e(k)
          e(k) = e(q)
          e(q) = xe
 42       xd   = d(k)
          d(k) = d(q)
          d(q) = xd
 43       xc   = c(k)
          c(k) = c(q)
          c(q) = xc
 44       xb   = b(k)
          b(k) = b(q)
          b(q) = xb
 45    continue
c
c Update q and search for another pair to interchange:
c
       q = q-1
       go to 20
 50    q = k-1
```

```
 60    continue
c
c The upwards search has now met the downwards search:
c
      a(i)=a(q)
      a(q)=ta
      go to (65,64,63,62,61,611,612,613),iring
 613      h(i) = h(q)
          h(q) = th
 612      g(i) = g(q)
          g(q) = tg
 611      f(i) = f(q)
          f(q) = tf
 61       e(i) = e(q)
          e(q) = te
 62       d(i) = d(q)
          d(q) = td
 63       c(i) = c(q)
          c(q) = tc
 64       b(i) = b(q)
          b(q) = tb
 65    continue
c
c The segment is now divided in three parts: (i,q-1),(q),(q+1,j)
c store the position of the largest segment in lt and ut
c
      if (2*q.le.i+j) go to 70
      lt(m) = i
      ut(m) = q-1
      i = q+1
      go to 80
 70   lt(m) = q+1
      ut(m) = j
      j = q-1
c
c Update m and split the new smaller segment
c
 80   m = m+1
      go to 10
c
c We arrive here if the segment has  two elements we test to see if
c the segment is properly ordered if not, we perform an interchange
c
 90   continue
      if (a(i).le.a(j)) go to 100
      xa=a(i)
      a(i)=a(j)
      a(j)=xa
      go to (95,94,93,92,91,911,912,913),iring
 913      xh   = h(i)
          h(i) = h(j)
          h(j) = xh
 912      xg   = g(i)
          g(i) = g(j)
          g(j) = xg
 911      xf   = f(i)
          f(i) = f(j)
          f(j) = xf
```

```
   91      xe    = e(i)
           e(i) = e(j)
           e(j) = xe
   92      xd    = d(i)
           d(i) = d(j)
           d(j) = xd
   93      xc    = c(i)
           c(i) = c(j)
           c(j) = xc
   94      xb    = b(i)
           b(i) = b(j)
           b(j) = xb
   95 continue
c
c If lt and ut contain more segments to be sorted repeat process:
c
 100  m = m-1
      if (m.le.0) go to 110
      i = lt(m)
      j = ut(m)
      go to 10
 110  continue
      return
      end
```

B.21 Subroutine SQDIST

```
      real*8 function sqdist(x1,y1,z1,x2,y2,z2,ind,MAXROT,rotmat)
c-----------------------------------------------------------------------
c
c     Squared Anisotropic Distance Calculation Given Matrix Indicator
c     ***************************************************************
c
c This routine calculates the anisotropic distance between two points
c  given the coordinates of each point and a definition of the
c  anisotropy.
c
c
c INPUT VARIABLES:
c
c   x1,y1,z1           Coordinates of first point
c   x2,y2,z2           Coordinates of second point
c   ind                The rotation matrix to use
c   MAXROT             The maximum number of rotation matrices dimensioned
c   rotmat             The rotation matrices
c
c
c
c OUTPUT VARIABLES:
c
c   sqdist             The squared distance accounting for the anisotropy
c                        and the rotation of coordinates (if any).
c
c
c NO EXTERNAL REFERENCES
c
c
```

```
c-----------------------------------------------------------------------
      real*8 rotmat(MAXROT,3,3),cont,dx,dy,dz
c
c Compute component distance vectors and the squared distance:
c
      dx = dble(x1 - x2)
      dy = dble(y1 - y2)
      dz = dble(z1 - z2)
      sqdist = 0.0
      do i=1,3
            cont   = rotmat(ind,i,1) * dx
     +             + rotmat(ind,i,2) * dy
     +             + rotmat(ind,i,3) * dz
            sqdist = sqdist + cont * cont
      end do
      return
      end
```

B.22 Subroutine srchsupr

```
      subroutine srchsupr(xloc,yloc,zloc,radsqd,irot,MAXROT,rotmat, &
     &                    nsbtosr,ixsbtosr,iysbtosr,izsbtosr,noct,nd, &
     &                    x,y,z,ddx,ddy,tmp,nisb,nxsup,xmnsup,xsizsup, &
     &                    nysup,ymnsup,ysizsup,nzsup,zmnsup,zsizsup, &
     &                    nsc,xs1,ys1,xs2,ys2,nclose,close,infoct)
!-----------------------------------------------------------------------
!
!              Search Within Super Block Search Limits
!              ***************************************
!
!
! This subroutine searches through all the data that have been tagged in
! the super block subroutine.  The close data are passed back in the
! index array "close".  An octant search is allowed.
!
!
!
! INPUT VARIABLES:
!
!   xloc,yloc,zloc   location of point being estimated/simulated
!   radsqd           squared search radius
!   irot             index of the rotation matrix for searching
!   MAXROT           size of rotation matrix arrays
!   rotmat           rotation matrices
!   nsbtosr          Number of super blocks to search
!   ixsbtosr         X offsets for super blocks to search
!   iysbtosr         Y offsets for super blocks to search
!   izsbtosr         Z offsets for super blocks to search
!   noct             If >0 then data will be partitioned into octants
!   nd               Number of data
!   x(nd)            X coordinates of the data
!   y(nd)            Y coordinates of the data
!   z(nd)            Z coordinates of the data
!   ddx(nd)          X difference for Kriging under BC
!   ddy(nd)          Y difference for Kriging under BC
!   tmp(nd)          Temporary storage to keep track of the squared
!                      distance associated with each data
```

```
!    nisb()                  Array with cumulative number of data in each
!                              super block.
!    nxsup,xmnsup,xsizsup    Definition of the X super block grid
!    nysup,ymnsup,ysizsup    Definition of the X super block grid
!    nzsup,zmnsup,zsizsup    Definition of the X super block grid
!    nsc                 Number of "gradient" segments
!    xs1(nsc)            X coordinates of the 1st endpoint of a "gradient" segment
!    xs2(nsc)            Y coordinates of the 1st endpoint of a "gradient" segment
!    ys1(nsc)            X coordinates of the 2nd endpoint of a "gradient" segment
!    ys2(nsc)            Y coordinates of the 2nd endpoint of a "gradient" segment
!
!
!
! OUTPUT VARIABLES:
!
!    nclose              Number of close data
!    close()             Index of close data
!    infoct              Number of informed octants (only computes if
!                          performing an octant search)
!
!
!
! EXTERNAL REFERENCES:
!
!    sqdist              Computes anisotropic squared distance
!    sortem              Sorts multiple arrays in ascending order
!    intersect           Checks if two segments are intersecting
!
!
!
!-----------------------------------------------------------------------
      real     x(*),y(*),z(*),ddx(*),ddy(*),tmp(*),close(*)
      real     xs1(*),ys1(*),xs2(*),ys2(*)
      real*8   rotmat(MAXROT,3,3),hsqd,sqdist
      integer  nsc
      integer  nisb(*),inoct(8)
      integer  ixsbtosr(*),iysbtosr(*),izsbtosr(*)
      logical  inflag, intersect
!
! Determine the super block location of point being estimated:
!
      call getindx(nxsup,xmnsup,xsizsup,xloc,ix,inflag)
      call getindx(nysup,ymnsup,ysizsup,yloc,iy,inflag)
      call getindx(nzsup,zmnsup,zsizsup,zloc,iz,inflag)
!
! Loop over all the possible Super Blocks:
!
      nclose = 0
      do 1 isup=1,nsbtosr
!
! Is this super block within the grid system:
!
            ixsup = ix + ixsbtosr(isup)
            iysup = iy + iysbtosr(isup)
            izsup = iz + izsbtosr(isup)
            if(ixsup.le.0.or.ixsup.gt.nxsup.or. &
     &         iysup.le.0.or.iysup.gt.nysup.or. &
     &         izsup.le.0.or.izsup.gt.nzsup) go to 1
```

```
!
! Figure out how many samples in this super block:
!
            ii = ixsup + (iysup-1)*nxsup + (izsup-1)*nxsup*nysup
            if(ii.eq.1) then
                  nums = nisb(ii)
                  i    = 0
            else
                  nums = nisb(ii) - nisb(ii-1)
                  i    = nisb(ii-1)
            endif
!
! Loop over all the data in this super block:
!
            do 2 ii=1,nums
                  i = i + 1
!
! Check if this is a "real" data point:
!
                  if((ddx(i).ne.0.0).or.(ddy(i).ne.0.0)) go to 2
!
! Check squared distance:
!
                  hsqd = sqdist(xloc,yloc,zloc,x(i),y(i),z(i),irot,MAXROT,
                                rotmat)
                  if(real(hsqd).gt.radsqd) go to 2
!
! Check if this point is screened by "no flow" segments:
!
                  do k=1,nsc
                     if(intersect(xloc,yloc,x(i),y(i),xs1(k),ys1(k), &
     &                   xs2(k),ys2(k)).eq..true.) go to 2
                  end do
!
! Accept this sample:
!
                  nclose = nclose + 1
                  close(nclose) = real(i)
                  tmp(nclose)   = real(hsqd)
 2          continue
 1    continue
!
! Sort the nearby samples by distance to point being estimated:
!
      call sortem(1,nclose,tmp,1,close,c,d,e,f,g,h)
!
! If we aren't doing an octant search then just return:
!
      if(noct.le.0) return
!
! PARTITION THE DATA INTO OCTANTS:
!
      do i=1,8
            inoct(i) = 0
      end do
!
! Now pick up the closest samples in each octant:
!
```

```
      nt = 8*noct
      na = 0
      do j=1,nclose
            i  = int(close(j))
            h  = tmp(j)
            dx = x(i) - xloc
            dy = y(i) - yloc
            dz = z(i) - zloc
            if(dz.lt.0.) go to 5
            iq=4
            if(dx.le.0.0 .and. dy.gt.0.0) iq=1
            if(dx.gt.0.0 .and. dy.ge.0.0) iq=2
            if(dx.lt.0.0 .and. dy.le.0.0) iq=3
            go to 6
 5          iq=8
            if(dx.le.0.0 .and. dy.gt.0.0) iq=5
            if(dx.gt.0.0 .and. dy.ge.0.0) iq=6
            if(dx.lt.0.0 .and. dy.le.0.0) iq=7
 6          continue
            inoct(iq) = inoct(iq) + 1
!
! Keep this sample if the maximum has not been exceeded:
!
            if(inoct(iq).le.noct) then
                  na = na + 1
                  close(na) = i
                  tmp(na)   = h
                  if(na.eq.nt) go to 7
            endif
      end do
!
! End of data selection. Compute number of informed octants and return:
!
 7    nclose = na
      infoct = 0
      do i=1,8
            if(inoct(i).gt.0) infoct = infoct + 1
      end do
!
! Finished:
!
      return
      end
```

B.23 Subroutine srchsupr2

Difference with *srchsupr* :
Instead of “Check if this is a "real" data point: (...)” :

```
!
! Check if this is a "gradient" data point:
!
                  if((ddx(i).eq.0.0).and.(ddy(i).eq.0.0)) go to 2
```

B.24 Subroutine SRCHSUPR3

Difference with *srchsupr* :
No “Check if this is a "real" data point: (...)”.
No “Check if this point is screened by "no flow" segments: (...)”.

Appendix C

GSLIB Code : Input Files

C.1 Example of a parameter file

```
                  Parameters for COKTBC
                  *********************

START OF PARAMETERS:
c:\Tests\K-BC2.txt
1                           \   number of variables primary+other
0                           \   estimated variable : 0=head, 1=transmittivity
1   2   0   3   4   5       \   columns for X,Y,Z,kod and variables
-9e+6   1e+29               \   trimming limits
0                           \co-located cokriging? (0=no, 1=yes)
                            \   file with gridded covariate
4                           \   column for covariate
3                           \debugging level: 0,1,2,3
coktbc.dbg
coktbc.out
51    0.0    10.0           \nx,xmn,xsiz
51    0.0    10.0           \ny,ymn,ysiz
1     0.5    1.0            \nz,zmn,zsiz
1     1      1              \x, y, and z block discretization
0                           \0=constant or 1=moving neighborhood
1     50     50     50      \min primary,max primary,max grad,max all sec
710   710    0              \maximum search radii: primary
710   710    0              \maximum search radii: all secondary
0     0      0              \angles for search ellipsoid
2                           \kriging type (0=SK, 1=OK, 2=OK-trad)
1 1 0 0 0 0 0 0 0 0         \drift: x,y,z,xx,yy,zz,xy,xz,zy
0.00   0.00   0.00   0.00   \mean(i),i=1,nvar
1                           \mean of transmissivity
0.1    270.0                \head gradient, gradient angle with X axis
0                           \grid file type
c:\Temp\tmp11B.tmp
0         1                    \use_log interpolation flag
0     0                     \Restrict_Min_Value, Value_Min
0     0                     \Restrict_Max_Value, Value_Max
1     1                     \semivariogram for "i" and "j"
1     0.01                     \   nst, nugget effect
6     1     0     0     0      \   it,cc,ang1,ang2,ang3
710   710   1                  \   a_hmax, a_hmin, a_vert
```

```
1     2                    \semivariogram for "i" and "j"
1     0.0                     \    nst, nugget effect
8     1     0     0     0     \    it,cc,ang1,ang2,ang3
710   710   1                 \    a_hmax, a_hmin, a_vert
2     2                    \semivariogram for "i" and "j"
1     0                       \    nst, nugget effect
1     1     0     0     0     \    it,cc,ang1,ang2,ang3
710   710   1                 \    a_hmax, a_hmin, a_vert
```

C.2 Example of a data file

```
Irregularly spaced 18 points
5
Column 1
Column 2
Column 3
Column 4
Column 5
395   25    0    2.07758    -9999999
155   85    0    6.534819   -9999999
315   145   0    8.481745   -9999999
55    185   0    12.45673   -9999999
415   215   0    10.37956   -9999999
245   245   0    13.05593   -9999999
375   305   0    13.10351   -9999999
35    315   0    19.32508   -9999999
195   375   0    18.22474   -9999999
305   415   0    16.15247   -9999999
455   455   0    14.53221   -9999999
65    485   0    28.41367   -9999999
0     500   0    50         -9999999
0     0     1    0          -9999999
500   0     1    0          -9999999
500   0.1   -1   0          -9999999
500   500   -1   0          -9999999
0.1   500   -1   0          -9999999
```